THE DISTRIBUTOR'S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A MATHEMATICAL FORMULATION AND HEURISTIC SOLUTION APPROACH

THESIS

Brian P. Ballew, Second Lieutenant, USAF

AFIT/GOR/ENS/00M-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE March 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE THE DISTRIBUTOR'S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A MATHEMATICAL FORMULATION AND HEURISTIC SOLUTION APPROACH	5. FUNDING NUMBERS
6. AUTHOR(S) Brian P. Ballew, Second Lieutenant, USAF	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GOR/ENS/00M-02
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dayton Area Graduate Studies Institute Attn: Dr. Frank Moore 3155 Research Blvd, Suite 205 Kettering, OH 45420 (937) 781-4000	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

JAMES T. MOORE, Lt Col, USAF (RET), AFIT/ENS e-mail: james.moore@afit.af.mil Phone: (937) 255-6565x4337

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	12b. DISTRIBUTION CODE

ABSTRACT (*Maximum 200 Words)*

It is through practice and experience Air Force loadmasters are able to pack the Air Force standard HCU-6/E (463L) pallets efficiently. Although the loadmasters perform their jobs exceptionally well, the Air Force is in search of a model that will more efficiently pack the pallets.

We have developed a mathematical formulation of the three-dimensional pallet-packing problem which minimizes the amount of unused space on a pallet. The formulation ensures each box is packed with the correct volume and dimensions, and ensures the volume of all the boxes packed is less than the available pallet volume. Additionally, the formulation ensures that each box has a foundation on which to be placed and allows, at most, one box to be placed in each location on the pallet.

The three-dimensional pallet-packing problem is a NP-hard problem. Thus, for large problems, the optimal solution can not be found in a reasonable amount of time. Therefore a heuristic solution approach is required to solve these large problems. This research observes the performance of a genetic algorithm on the three-dimensional pallet-packing problem using single-point crossover.

14. SUBJECT TERMS Three-dimensional pallet-packing, packing problems, pallet-loading problems, distributor's pallet-packing problem, heuristics, genetic algorithms	15. NUMBER OF PAGES 111
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNC

AFIT/GOR/ENS/00M-02

THE DISTRIBUTOR'S THREE-DIMENSIONAL PALLET-PACKING PROBLEM: A MATHEMATICAL FORMULATION AND HEURISTIC SOLUTION APPROACH

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science

Brian P. Ballew, B.S.

Second Lieutenant, USAF

March 2000

AFIT/GOR/ENS/00M-02

THESIS APPROVAL

Student: Brian P. Ballew, Second Lieutenant, USAF   Class: GOR-00M

Title: The Distributor's Three-Dimensional Pallet-Packing Problem: A Mathematical Formulation and Heuristic Solution Approach

Defense Date: 02 March 2000

Committee:	Name/Title/Department	Signature
Advisor	James Moore, Ph. D. Associate Professor Department of Operational Sciences	James T. Moore
Reader	Raymond Hill, Major, USAF Assistant Professor Department of Operational Sciences	Raymond R Hill
Reader	Gregory McIntyre, Lt. Colonel, USAF Assistant Professor Department of Operational Sciences	[illegible]

Acknowledgements

First, I would like to thank my thesis advisor, Dr. Moore, for his support and guidance throughout this thesis effort. I would also like to thank one of my readers, Major Hill, for his insight on the problem and his help with the establishment of a sponsor. I would also like to thank Lt Col McIntyre for his help with GENESIS and genetic algorithms.

My family and my girlfriend, Betsy, deserve a lot of thanks for putting up with me during these trying months and supporting me throughout the thesis effort. My parents deserve extra thanks for teaching me the commitment and discipline needed to complete this program.

Lastly, I would like to thank my classmates for all of their help throughout the program and the good times.

Table of Contents

List of Figures

Figure Page

List of Tables

AFIT/GOR/ENS/00M-02

Abstract

It is through practice and experience Air Force loadmasters are able to pack the Air Force standard HCU-6/E (463L) pallets efficiently. Although the loadmasters perform their jobs exceptionally well, the Air Force is in search of a model that will more efficiently pack the pallets.

We have developed a mathematical formulation of the three-dimensional pallet-packing problem which minimizes the amount of unused space on a pallet. The formulation ensures each box is packed with the correct volume and dimensions, and ensures the volume of all the boxes packed is less than the available pallet volume. Additionally, the formulation ensures that each box has a foundation on which to be placed and allows, at most, one box to be placed in each location on the pallet.

The three-dimensional pallet-packing problem is a NP-hard problem. Thus, for large problems, the optimal solution can not be found in a reasonable amount of time. Therefore a heuristic solution approach is required to solve these large problems. This research observes the performance of a genetic algorithm on the three-dimensional pallet-packing problem using single-point crossover.

Chapter 1 – Introduction

1.1 Background

The mission of AMC is "The Air Mobility Team … Responsive Global Reach for America … Every Day." An important aspect of this responsiveness is providing the troops in forward locations with the goods they need using the fewest number of planes. Therefore, in hope of optimizing the use of our aircraft while simultaneously minimizing transportation costs, loadmasters attempt to maximize the amount of cargo on a given aircraft. Another aspect of cargo loading that could help cut costs and increase airlift efficiency is the packing of cargo on the pallets.

Pallets are packed by Air Force loadmasters. Loadmasters attend a training course on the basic principles of pallet packing, yet it is through practice and experience that they are able to pack the pallets as efficiently as they do. Experienced loadmasters eyeball the items to be loaded and determine the best way to load them. The pallets are Air Force standard HCU-6/E (463L) pallets. The length and width of the pallets are 88 inches (7 feet 4 inches) and 108 inches (9 feet), respectively. However, only 84 inches of the length and 104 inches of the width are available to be packed. Loadmasters are required to leave the outside two inches of the pallet unpacked so that the cargo net is able to securely fit around the packed boxes. The maximum height of a pallet is 96 inches (8 feet) for pallets loaded in the main compartment and 76 inches (6 feet 4 inches) for pallets loaded on the ramp (Taylor, 1994).

Although loadmasters perform their jobs exceptionally well, the Air Force is in search of a model that will help efficiently pack the pallets and provide loadmasters with a report stating where the boxes should be placed on the pallet to minimize unused space. This will save time and resources. If the pallets are more efficiently packed, the number of sorties flown may decrease and aircraft may be freed to carry other items in large-scale mobilizations.

The Air Force has sponsored research in this area on multiple occasions in search of finding a better way to pack the pallets and load the aircraft. These include an early effort by Taylor (1994), an airlift-loading model by Chocolaad (1998), and a three-dimensional packing problem approach by Manship and Tilley (1998).

Taylor developed three different models in an attempt to solve the hybrid two-dimensional packing problem. The goal of the hybrid packing technique is to pack the boxes in layers as in the two-dimensional case. An additional goal is to minimize the deviation in height between the boxes in each layer. This helps make the height of each layer as uniform as possible (Taylor, 1994).

Taylor's first model minimized the deviation in height between the boxes in a layer while maximizing the area covered. In his second model, the objective was to minimize the deviation in volume wasted based on whether or not a given box was loaded. The amount of wasted volume depended on whether or not any unused surface area of the pallet remained due to choosing a specific box. By minimizing the largest of these values over the set of boxes, the most volume efficient packing solution could be found (Taylor, 1994). Taylor's third model was an extension of the second model, and the objective was to minimize the wasted volume of all packed boxes.

Taylor was able to come up with optimal solutions for his model, but due to limited computer resources and capabilities, he was only able to solve three very small problems (Taylor, 1994). The first problem was loading four boxes onto a three by three pallet. The second problem was loading seven boxes onto a four by four pallet in which one box was manually placed. The last problem was loading eight boxes onto a five by five pallet in which two boxes were manually placed. Since his test problems were only concerned with packing two layers on a pallet, the pallets did not have a height dimension.

Chocolaad's (1998) research pertains to the airlift-loading problem. In his research, he developed a packing heuristic which uses simple tabu thresholding to determine the packing. This is a simple search method which avoids becoming trapped at a local optimum by allowing non-improving moves (Chocolaad, 1998). However, Chocolaad's research addresses the airlift loading problem and not the pallet-packing problem. Thus, he is trying to maximize utilization of the aircraft as opposed to the utilization of a pallet. In the two-dimensional case, the airlift-loading problem is similar to the pallet-packing problem in that the goal is to maximize the number of different size items (boxes/cargo) to fit into an airplane or pack onto a pallet subject to some constraining factors.

Manship and Tilley (1998) approached the three-dimensional packing problem using a nonlinear programming approach. Unfortunately, their pallet-packing model only provides sub-optimal solutions. However, it does provide feasible solutions that include most of the required constraints (Manship and Tilley, 1998).

Efforts in the past to improve on current pallet-packing procedures have at best only produced marginally better pallet loads, yet due to the complexity of the problem, very little if any time is saved due to computational times (Taylor, 1994). Therefore, there is much room for improvement in the area of pallet packing. A model that can efficiently pack pallets not only will cut costs but will free time and resources to be used elsewhere.

1.2 Three-Dimensional Pallet-Packing

Of all the research devoted to optimization problems, only a very small percentage of that research focuses on packing problems, and even a smaller percentage of that research concentrates on three-dimensional pallet-packing problems. The main reason for this is that if the three-dimensional pallet-packing problem is formulated as an integer program, it becomes an extremely complex and difficult to solve problem. As the pallet dimensions increase and the number of boxes to be packed increase, the number of variables required to model the problem explodes; thus, the problem does not admit a solution-time based on a polynomial function of the problem size.

Although the focus of this research is in three-dimensional pallet packing, it is important to describe how the three-dimensional pallet-packing problem differs from the other types of packing problems. First, there are two general types of packing problems; there are the 'manufacturer's pallet packing problem' and the 'distributor's pallet packing problem'. The 'manufacturer's pallet packing problem', also referred to as the pallet loading problem, is the problem of finding the optimal layout for identical rectangular boxes on a rectangular pallet (K. Dowsland, 1987). Generally, the pallet is packed in layers where the second layer is packed the same as the first layer and so on until the height constraint of the pallet is reached. Thus, the formulation for this problem is reduced to the two-dimensional case and the objective is to maximize the number of boxes packed on the pallet.

For the 'distributor's pallet packing problem,' the objective is to load boxes of varying dimensions onto as few pallets as possible (Askin and Standridge, 1993). This is a more difficult problem. For the case in which only one pallet is to be loaded, the objective is to maximize volume utilization (minimize the amount of unused space). Most work done with the three-dimensional case of this problem has attempted to group boxes with the same or similar height to form layers. Then using these layers, the algorithm packs the layers until the height constraint is reached.

Other packing problems very similar to the three-dimensional pallet-packing problem are the three-dimensional container-loading problem and the bin-packing problem. For all three of these problem types, the objective is to maximize volume utilization. However, the container-loading problem and the bin-packing problem have one less constraint than the three-dimensional pallet-packing problem. Since both bins and containers have vertical walls providing load stability, neither of these two problem types includes a load stability constraint. Load stability means the boxes do not have the tendency to tip over once packed. The three-dimensional pallet-packing problem on the other hand must include the stability constraint to ensure the packing of the boxes is stable.

The distributor's three-dimensional pallet-packing problem is the most difficult of the packing problems. Unfortunately, expanding two-dimensional formulations to incorporate the third dimension becomes intractable for realistic problems. In addition, it is very difficult to modify these formulations to meet the additional required constraints of the three-dimensional pallet-packing problem, particularly the stability constraint.

1.3 Statement of the Problem

The purpose of this research is to develop a three-dimensional pallet-packing algorithm that can be adopted by the Air Force to improve the current packing procedure. Few models have been developed to attack the three-dimensional palletization problem with non-uniform box sizes, and as mentioned in the previous section, not many of these models took a strict three-dimensional approach. Most algorithms attack the third dimension using some pseudo three-dimensional approach, such as the layered approach. This research employs a strict three-dimensional procedure for non-uniform box sizes that more efficiently packs pallets.

1.4 Restrictions and Assumptions

Due to the complexity of the three-dimensional palletization problem as well as the lack of previous work that has been accomplished in this area, many simplifying assumptions are employed in the formulation. The goal is to start simple and add complexity until we can pack Air Force 463L pallets.

The first two restrictions are quite obvious. The first restriction is that only one box can occupy each pallet location. The second restriction is that the boxes are not permitted to extend beyond the dimensions of the pallet. Thus, the volume of all packed boxes must be less than the available pallet volume. Before a pallet is loaded onto a plane, the loadmaster secures the pallet by tying down cargo nets around the load.

Protruding boxes prevent the loadmaster from adequately securing the load. Therefore, to help the formulation "pack the boxes" so that they fit within the boundaries of the pallet, each box can be packed in any of its six orientations.

Additionally, we assume in the formulation that no overhang is allowed when the boxes are packed. For a box to be packed, its entire base must be on top of either the pallet or other boxes. A further restriction is that all boxes are rectangular in nature. Both of these restrictions are commonly found in three-dimensional formulations. The first prevents the packed boxes from tipping, which adds to the stability of the load, whereas the second restriction simplifies the problem so spherical and cylindrical objects do not have to be accounted for in the formulation.

Generally, Air Force 463L pallets are "cubed out" before they are "grossed out" (Taylor, 1994). This means the total available volume of the pallet is generally filled before the allowable weight limit is reached. For this reason, the weight of the boxes is initially omitted from the formulation. Therefore, all constraints dealing with weight are omitted.

The first weight constraint omitted is the constraint ensuring the weight of all packed boxes does not exceed the available weight of the pallet. The second weight constraint omitted ensures heavier boxes are placed below lighter boxes. The third weight constraint omitted is the center of gravity constraint. For safety reasons, the Air Force requires that the center of gravity of a load is within four inches of the center of the pallet.

Another constraint not included in this formulation is the load stability constraint. This constraint is omitted since each box is required to have a complete foundation under it. Additionally, we are omitting the constraint which ensures the top of the load is as close to level as possible. This is required so that when the cargo net is thrown over the load, boxes do not shift or fall. Lastly, we are only concerned with packing boxes onto one pallet. These assumptions aid in the development of a formulation strictly concerned with the placement of boxes; yet allows for future modifications increasing the realistic nature of the formulation.

1.5 Overview

Chapter Two presents a detailed review of past work accomplished in this field of study and describes some of the solution techniques used by others. In addition, it touches upon a couple of commercial three-dimensional pallet-packing software packages available for purchase. Chapter Three describes in detail the development of the formulation and the constraints in the formulation. Since for larger, more realistic problems the number of variables increases exponentially, we also discuss in Chapter Three the heuristic applied to solve the problem's formulation. Lastly, Chapter Four provides a formal conclusion and recommendations for follow-on research.

Chapter 2 – Literature Review

2.1 Introduction

Much of the previous work on packing problems pertains to the two-dimensional packing problem. Yet in recent years with the advancements in computer technology and the advent of new solution techniques, there has been an influx of research and published papers in the realm of the three-dimensional pallet-packing problem. Unfortunately, the three-dimensional pallet-packing problem is a NP-hard problem; thus, for relatively large problems, the optimal solution can not be found in a reasonable amount of time. For this reason, much of the research on the three-dimensional palletization problem includes the implementation of a heuristic to find a good solution.

"A heuristic is a technique which seeks good (i.e., near optimal) solutions at a reasonable computational cost without being able to guarantee either feasibility, or optimality, or even in many cases to state how close to optimality a particular feasible solution is" (Reeves, 1995). Instead of having to search the entire solution space and enumerate all possible solutions to find the optimal solution, a heuristic provides a means for smartly searching the solution space, examining only those areas where the optimal solution most likely resides. The heuristic will not necessarily converge on the optimal solution; yet, they generally provide solutions close to optimal in a reasonable amount of time.

The remainder of this chapter reviews past research pertaining to the three-dimensional pallet-packing problem. Additionally, it reviews a couple of the commercial pallet-packing software packages available for purchase. Lastly, it reviews the application of heuristics to the three-dimensional pallet-packing problem, specifically the application of the genetic algorithm to solving the three-dimensional palletization problem.

2.2 Three-Dimensional Pallet-Packing Problem

For nearly twenty years both Kathryn A. Dowsland and William B. Dowsland have been researching packing problems. Most of their earlier work focuses on the two-dimensional pallet-packing problem, yet they do address the three-dimensional packing problem. In a combined packing overview, one paper suggests that, on average, a three-dimensional algorithm will better load boxes on a pallet. However, they state this will come at an expensive computational cost and there will always be situations when a manual packing will more efficiently pack a pallet (Dowsland and Dowsland, 1992).

William Dowsland (1991) states that since this is such a new field, much of the published work regarding the three-dimensional palletization problem declares successful implementation, but fails to provide the reader with any clear measure of scientific success. Since this is such a new area of study and many different methods are being attempted, any work illustrating a successful implementation of an algorithm is published. Scientific results stating how well a particular method performed on the three-dimensional pallet-packing problem are generally not provided, yet this is the information most useful to those performing follow-on research as it dictates what methods to avoid, or what methods to pursue further.

Dowsland addresses many of the methods used by researchers to pack pallets in the third dimension. Unfortunately, he states many of these methods will become intractable when extended to larger, more realistic problems (Dowsland 1991). Two methods discussed are the layered approach and the 'L' packing approach. The idea behind the layered approach is grouping boxes of the same height together and packing that group along the bottom of the pallet. Boxes of the same height are continually grouped and then packed as the next layer on the pallet. This continues until the height constraint is reached. If there are not enough boxes of the same height to fill a layer, then the algorithm searches for boxes closest in height and packs them as a layer.

The problem with using the layered approach when packing in three dimensions is accounting for the stability of the load and the weight of the boxes. Moreover, if there are many boxes with different heights, the packing efficiency drops. It is generally good practice to first pack the larger, heavier boxes on the pallet to provide a foundation for the other boxes. However, a layered approach will pack the boxes with the same height first no matter their weight or volume. Thus, when using the layered approach in three dimensions, there are numerous additional checks the algorithm must perform to ensure the packing satisfies these additional constraints.

The 'L' approach is also known as the wall building approach. As in the layered approach, we first want to group boxes with the same height and pack them as a layer along the floor of the pallet. Then, boxes with similar dimensions are grouped together and stacked vertically along one of the walls until the height constraint is reached. The algorithm continues alternating between packing the boxes in a horizontal layer and a vertical column until there are no more boxes that can be packed or there are no more boxes to pack. Han, Knott, and Egbelu (1989) applied this approach to the manufacturers pallet-packing problem and were able to achieve good results.

When packing different sized boxes, the main concern with this type of packing is how to ensure packing stability since this type of packing technique can produce gaps between the vertical columns of packed boxes (Dowsland, 1991). Additionally, when there are not enough boxes to completely fill the pallet, the algorithm produces unbalanced loads. Lastly, it is preferred to pack pallets upward from the base of the pallet. For these reasons, the 'L' approach has been known to produce loads that do not satisfy the stability constraint nor the center of gravity constraint, and thus this approach has not been used except for cases where the boxes to be packed are of uniform size.

Abdou and Yang (1994) developed a multi-objective algorithm for the palletization problem. They attempt to maximize both pallet utilization and stability. Stability is needed to prevent the boxes from toppling when packed. To prevent this, larger, heavier boxes should be packed below the smaller, lighter boxes. This provides a solid foundation for the next layer of boxes. In their approach, Abdou and Yang attempt to simplify the problem by grouping boxes with the same height.

In addition, the authors develop blocks that are subsets of the actual pallet. They try to maximize the space filled within each block, by either filling it with a large box or with several smaller boxes. As expected, when new blocks are generated, the criteria is to use boxes that have not been previously selected, use the bigger boxes first, and lastly use as few boxes as possible. This makes it easier to pack the next layer. By packing the larger boxes first, either the volume of the block will be filled, or the larger boxes provide a good foundation for the boxes placed above them.

Abdou and Yang's algorithm has many features required by the Air Force. However, to increase the capability of this model, boxes with similar heights but different base dimensions should not be grouped together. Abdou and Yang's current model uses a layered approach to attack the third dimension. Although the boxes are not the same size, they are grouped by height so that the packing can be performed in layers. By disallowing this requirement, the complexity of the problem increases, however the opportunity for a better solution increases as well.

Abdou and Arghavani (1997) developed a three-dimensional algorithm that arrives at its solutions through two separate objective functions. The first objective function attempts to maximize the packed base area of the pallet. Additionally, sub-areas are defined within this objective function. The second objective function seeks to maximize the stacking column for each sub-area.

Abdou and Arghavani (1997) discuss some constraints for their three-dimensional pallet-packing problem. One constraint ensures all boxes are packed within the confines of the pallet. Another ensures the stacking heights are less than or equal to the maximum pallet height. A third constraint limits the number of each type of box available while a fourth constraint allows the boxes to be packed in only one of two orientations. This is also known as the 'face up' constraint where the boxes can only rotate 90° around the vertical axis.

Ivancic, Mathur, and Mohanty (1989) attack the three-dimensional packing problem to minimize the number of pallets required to hold all the boxes. This is basically the same as maximizing pallet efficiency. The authors formulate the problem as a multidimensional knapsack problem and use a greedy heuristic to pack the boxes. Instead of packing boxes by using the "biggest bang for the buck" philosophy, their algorithm favors packing the box type having the most boxes left unpacked or those with smaller volumes. The algorithm avoids packing boxes that trap unpacked space. This algorithm does allow the boxes to be packed in any of the six orientations, yet a major drawback of this heuristic is that it requires knowledge of all the packing patterns, which for large problems is computationally prohibitive (Ivancic, Mathur, and Mohanty, 1989).

Bischoff, Janetz, and Ratcliff (1995) developed a three-dimensional heuristic approach to packing multiple-sized boxes onto a pallet, also known as the "Distributor's Pallet Packing Problem." Their objective function maximizes pallet utilization while ensuring the load is stable. The algorithm developed by the authors packs the boxes in layers allowing up to two different box types per layer; however, it gives preference to those layers which can be filled by a single box type.

The authors found the layering approach provided better stability than those algorithms which pack boxes in vertical columns. Additionally, the algorithm prefers larger gaps between boxes to smaller ones. The larger the gap, the better the chance a box will be able to fill the vacancy. The algorithm did produce stable loads, but as the number of various sized boxes increased, the packing efficiency declined.

Bischoff and Ratcliff (1995) published a follow-on to the article described above. They packed pallets in layers allowing at most two different box sizes per layer. The focus of this research was to determine whether it is better to load multiple pallets simultaneously or pack them sequentially. Simultaneous packing means that all the pallets are loaded at the same time, whereas sequential packing means that one pallet is fully loaded before any boxes are placed on the next pallet. The authors found packing the pallets sequentially to be the better packing method.

Fuh-Hwa and Hsiao (1997) address the three-dimensional "manufacturer's pallet loading problem." Since all the boxes are the same size, the authors use a layered approach to pack the boxes. The boxes are packed with any orientation; however, they must be packed so that the height of each layer is constant. While a primary objective is to maximize the number of boxes on each pallet, stability is also important. They check to ensure no packed vertical columns are removed from the rest of the packing. They also ensure each box has a solid base underneath it. Lastly, for problems where the boxes are not all the same size, they pack the heavier, larger boxes on the bottom of the pallet providing a solid foundation for the remaining boxes.

Fuh-Hwa and Hsiao have developed a three-dimensional heuristic to maximize the pallet load. They model this problem with boxes of four different sizes arriving randomly at a loading dock. Once at the loading dock, the critical path method is used to determine the placement and sequence of the boxes. For example, for a box to be packed, there must be a solid, uniform foundation for that box. Although the technique may be transferable to the Air Force's pallet-packing problem, the scenario is different. The authors developed this model to pack boxes arriving on a conveyor belt as opposed to packing boxes already at the loading site.

Tsai, Malstrom, and Kuo (1993) developed an exact model for the three-dimensional pallet-packing problem. Unfortunately, as the number of boxes increases, the computation time required to find the optimal solution increases and limits the practical use of this model. However, the authors do provide the constraints required to ensure no two boxes overlap and that all boxes, if packed, are packed within the confines of the pallet.

The authors' model did not include the constraints that address the issue of load stability. The model was designed for the single pallet case and thus the objective of the model was to maximize the packing volume. They did not consider the case when there are multiple pallets to be packed. Lastly, the authors did allow the boxes to be packed in one of two orientations; the boxes were allowed to rotate 90° around the vertical axis.

Tsai, Malstrom, and Kuo introduced a heuristic to solve the three-dimensional pallet-packing problem. This paper focuses on packing boxes arriving via a conveyor belt. Additionally, this problem is concerned with packing multiple pallets simultaneously. The arrival of different sized boxes on a conveyor belt does reflect how boxes arrive in distribution centers, however it is quite different from this research's problem.

2.3 Previously Contracted Efforts

The Air Force has been examining this problem for many years in an attempt to minimize the costs associated with shipping cargo. In addition to the past research efforts at the Air Force Institute of Technology (AFIT), the Air Force has contracted this problem to companies outside the DoD. Both the Computer Science Corporation (1997) and TASC, Inc. (1998) performed research on this topic.

The Computer Sciences Corporation's research states that although a strict mathematical model which produces optimal loads may not be available, heuristic techniques are available which could enhance load efficiency as well as reduce the time it takes to produce a load (Computer Sciences Corporation, 1997). Thus, they do feel a software package can be developed which takes into account the necessary packing constraints and is able to find a close to optimal packing. Additionally, they state it is crucial for the software package to provide instructions describing exactly how to achieve the packing found by the software.

The research provided a review of current systems employed by the Air Force and a review of existing literature on the pallet-packing problem. Unfortunately, none of the existing systems used by the Air Force specifically address packing pallets. Therefore, the research reviewed existing literature on the packing problems and commentated on how the current research pertained to the Air Force's requirements.

TASC, Inc. researched the three-dimensional pallet-packing problem. The initial focus of their research was the feasibility of finding optimal solutions. However, as they became more familiar with the problem and its inherent complexity, the focus of the research changed to the feasibility of finding reasonably good solutions (TASC, 1998). Before they looked at solving the problem, they performed a literature search on past pallet-packing efforts. They found a lot of research has been performed on this problem, yet none of the research was complete in the sense that it encompassed all the requirements of the Air Force.

The TASC research provided a list of all the packing rules and constraints used by the Air Force. While some of the constraints are required, other constraints are either recommended or optional.

Before introducing the pallet-packing software they developed, TASC described a few of the existing pallet-packing software packages. The software package they developed was LoadPal (short for load pallet) (TASC, 1998). This program was written in C++ and contained most of the major constraints. Unfortunately due to time constraints, the program did not include a Graphical User Interface (GUI).

After testing their software package, they found that their approach was comparable to other existing software packages. However, they stated that their package could be expanded to include the other required constraints and employ a better solution technique which would provide statistically better solutions than those found from existing software packages. They suggest the cost of developing this product would be $150,000 and it would require six months of effort.

2.4 Commercial Packing Software

The Remarkable Software Company located in New Zealand is selling a three-dimensional pallet-packing software package (PowerPak^{TM}) that may possibly be of use to the Air Force. They have developed PowerPak^{TM} to allow boxes with different sizes and attributes be loaded into the program. There is also no limitation on the size of the pallet or the size of the boxes as long as each uses the same units. Additionally, there is no limit on the number of boxes that can be input into the system. However, the program only allows the user to pack one pallet at a time.

PowerPak^{TM} also allows the user to input the maximum allowable weight for the pallet as well as the weight of each box. Using these values, the program determines the packing of boxes onto the pallet to minimize empty space. Once a solution is found, the software allows the user to rotate the pallet to see where each box was placed, but more importantly it provides an Optimized Packing Sequence Report. This report allows the user to achieve the computed packing through a step by step process.

Unfortunately, there is little insight into which technique(s) PowerPak^{TM} uses to pack boxes on a pallet. The product does take weight into account, making sure the weight of all packed boxes does not exceed the allowable pallet weight, but it is unclear whether or not the model ensures heavier boxes are packed towards the bottom.

Another commercial software product pertaining to pallet loading is Capesystems. Capesystems allows the user to pack up to three pallets simultaneously. Moreover, this product does not limit the shapes of the items to be packed. It packs cases, cylinders, ovals, and trapezoidal items. Three different modules are available to deal with the range of pallet-packing problems.

The first module is Single Product and is used for the loading of identically sized boxes. For a large problem, this module's computation time is around 15 seconds. The next module, Display Pallet, allows the user to load products of up to 40 different sizes. However, this module has a limit on pallet dimensions. Each dimension must be less than 100 inches. Therefore, since the length of the 463L pallet is 108 inches wide, this module would not meet Air Force requirements. Depending on the number of different sized boxes to be packed, the solution time for this module is around one minute. For both of these products, the program uses a series of logistical algorithms to pack the boxes in layers or in columns.

The last module is concerned with packing products with different dimensions perpendicular to the pallet. This term means that the items packed must be packed upright. Since this research is not concerned with non-rectangular shapes, little information was gathered on this module.

However, they do have another product, Truckfill, which handles the larger sized pallets such as the 463L pallet. This product was designed more for logistics than it was for optimization, thus the packing is not as efficient as one would get using either of the other two modules. Since this module allows for such a diverse range in problem size, Capesystems provided no estimate regarding the average solution time using Truckfill.

2.5 Heuristics

Heuristics are commonly applied to packing problems so that realistically large problems can be solved in a reasonable amount of time. Although the solutions are not guaranteed to be optimal or even close to optimal, heuristics generally provide good solutions. A variety of heuristics have been applied to the three-dimensional pallet-packing problem, but the three most common heuristics used are the greedy heuristic, simulated annealing and genetic algorithms.

2.5.1 Greedy Heuristic

A greedy heuristic is a simple approach to the packing of the boxes on a pallet. The algorithm packs the largest boxes first and continues down the list of boxes until no more boxes can fit on the pallet. Unfortunately, the greedy heuristic does not necessarily provide efficient loads and thus is not used except to provide starting solutions for other heuristic techniques.

2.5.2 Simulated Annealing

Simulated annealing is another heuristic that has been utilized in packing problems. K. Dowsland (1993) used simulated annealing techniques on the two-dimensional packing problem. Simulated annealing is a probabilistic search technique in that moves are selected with some associated probability.

For the packing problem, simulated annealing begins with some initial solution. This solution can be found through the use of a greedy heuristic or it can be as simple as starting with an empty pallet. The algorithm then proceeds through the list of boxes and picks a box at random to enter the packing. Entering that box into the packing is considered a "move." If that box increases pallet utilization while maintaining a feasible solution, then the move is accepted. However, if the move is infeasible, then a box is picked at random to be removed from the solution. All moves that increase pallet utilization are accepted as long as feasibility is maintained. However, if a move decreases pallet utilization, the move is only accepted according to some probability function designed by the user.

In the initial stages of the algorithm, it is generally good practice to have the algorithm accept both improving and non-improving moves. This prevents the search from converging on a local optimum. Then as the search proceeds, the algorithm should select fewer unimproving moves to encourage the algorithm to converge on the global optimum. This is referred to as the cooling schedule. It is defined by the user and is generally different for each problem. Defining a cooling schedule is a very difficult and timely process. "It is said that while simulated annealing is easy to get working it is difficult to get working well." (Dowsland, 1993)

2.5.3 Genetic Algorithms

A third heuristic technique most commonly applied to pallet-packing problems is genetic algorithms. Genetic algorithms exploit historical information to speculate on new search points with expected improved performance. Genetic algorithms are based on the ideas of natural selection and the goal is to start with a group of initial solutions, called a population, and use these initial solutions to guide the search to the optimal solution. The initial population can be randomly generated or it can be initialized using solutions from quick, greedy heuristic techniques. The initial population is referred to as the first generation. After each iteration, a new population is produced. The user must determine the number of iterations to perform or the number of generations to generate before terminating the search.

Additionally, the user must determine the size of the population. This is a difficult parameter to assess since it varies for each problem type and problem size. If a population size is too small then there is an insufficient number of different solutions, the population is not diversified enough, and the genetic algorithms may converge on sub-optimal regions. On the other hand, an extremely large population provides too much diversification and genetic algorithms may fail to converge to even local optimal solutions. Additionally, with extremely large populations, the solution time increases drastically. Therefore, the goal is to find a population size which provides enough diversification with reasonable solution times to obtain good solutions.

Once a population is established, members from that population are combined (mated) to produce new solutions (a new generation). Four genetic algorithm parameters determine which members of a population should mate, how they should mate, and which candidate solutions should become the members of the next generation. These parameters are the selection strategy, crossover rate, mutation rate, and generation gap.

The selection strategy determines which members of a population mate. Proportional selection, the elitist strategy, and rank-based selection are three of the more popular selection strategies. In each method, members are randomly selected, however the random selection differs for each strategy. Proportional selection is the most common of all selection strategies. This strategy proportionally assigns probabilities to candidate solutions based on their objective function value. Those solutions with better objective function values have a higher probability of being selected for mating. The elitist strategy is a variation of proportional selection that ensures the best candidate solution survives into the next generation (Grefenstette, 1990). The elitist strategy generally converges on solutions quicker than a completely random selection; however, it sometimes converges prematurely on sub-optimal locations. The ranking strategy ranks the candidate solutions and assigns probabilities based on their rank within the population. Thus, higher-ranking solutions have a greater probability of being selected for mating.

Two basic operations are responsible for how the actual mating occurs. They are crossover and mutation. The idea behind crossover is that of exploitation. The goal of exploitation is to mate two parents associated with good solutions to produce offspring which have an improved solution. This continues until the genetic algorithm converges on a good solution or is kicked out of that region. Crossover combines selected portions of one parent with selected portions of the second parent to produce an offspring.

Generally, each solution is represented by a bit stream. Figure 2-1 provides an example of single-point crossover.

Parent 1:	001001 || 1101	Child 1:	001001 || 1001
Parent 2:	010101 || 1001	Child 2:	010101 || 1101

Figure 2-1: Single-point Crossover

There are numerous types of crossover operations that can be applied in a genetic algorithm. The most primitive type of crossover operation is the single-point crossover. The algorithm randomly chooses a place along the bit stream where the crossover should occur. In Figure 2-1, the vertical lines after the sixth bit signify the crossover point. Once this crossover point has been established, the algorithm takes the first group of bits from the first parent and combines them with the second group of bits from the other parent to produce a new offspring or child. Thus for Child 1, the first six bits come from the first six bits of Parent 1, while the remaining four bits come from the last four bits of Parent 2. For Child 2, the first six bits come from the first six bits of Parent 2, while the remaining four bits come from the last four bits of Parent 1.

Although single-point crossover is the simplest type of crossover, it is not necessarily the best type of crossover. In fact, single-point crossover limits the amount of information which is exchanged between the parents (Reeves, 1995). To combat this, multi-point crossover can be applied to improve the performance of a genetic algorithm.

Another type of crossover that can be applied is 'string-of-change' crossover. (Reeves, 1995) This is a special type of single-point crossover that should be applied only when the bit stream of both parents to be mated is similar through the first few bits. This is because if both bit streams are similar through the first few bits and the single-point crossover occurs in this location, then the crossover will fail to produce a new string. Thus, 'string-of-change' crossover checks the bit streams to ensure the crossover point produces offspring with different bit streams than the parents.

A third type of crossover is referred to as the generalized crossover, also called uniform crossover (Reeves, 1995). For this type of crossover, after the parent bit streams have been chosen, a third bit stream of equal length to the other bit streams is randomly generated. Figure 2-2 below provides an example of generalized crossover.

Parent 1:	0010011101	Child 1:	0111011101
Parent 2:	0101011001	Child 2:	0000011001
Random Bit Stream:	1010010110		

Figure 2-2: Generalized Crossover

For Child 1, wherever there is a '1' in the random bit stream, the corresponding bit is taken from Parent 1, and wherever there is a '0' in the random bit stream, the corresponding bit is taken from Parent 2. Thus, the random bit stream provided in Figure 2-2 implies that for Child 1, the 1st, 3rd, 6th, 8th, and 9th elements are taken from Parent 1, while the 2nd, 4th, 5th, 7th, and 10th elements are taken from Parent 2. It is the opposite for Child 2. Wherever there is a '1', the element is taken from Parent 2, and wherever there is a '0', the element is taken from Parent 1. These three types of crossover are the most common types and have been successfully implemented in past research.

Mutation attempts to jump the solution out of the current solution space by adding exploration to the search. The purpose of mutation is to prevent the genetic algorithm from quickly converging on a sub-optimal solution and is applied after the crossover has taken place. Mutation occurs when the value of a single bit in a solution is flipped. For example, if a bit chosen for mutation has a value of 1, after mutation its value is changed to 0. The user defines how much mutation to include in the problem, but generally the mutation rate is less than five percent. Each bit in the bit stream has an equal probability of being mutated. Therefore, based on the probability introduced by the user, the algorithm determines which bit(s) to mutate.

The generation gap determines the percentage of offspring to include in the next generation. There are two methods used to determine this; the generational genetic algorithm and the incremental genetic algorithm. In the generational genetic algorithm, all offspring are included in the next generation, completely replacing the previous generation. In the incremental genetic algorithm, only selected offspring are included in the next generation. Generally, they replace those members from the previous generation associated with the worst solutions.

It is generally good practice to replace a majority of the parent generation, yet it is also important to allow some of the parent generation to survive from one generation to the next. When only a small portion of the parent generation is replaced, the algorithm frequently fails to converge on a feasible solution. On the other hand if we replace the entire parent generation, then it is possible that we are removing good candidate solutions from our solution space. However, most genetic algorithms maintain a list of the best solutions found to date. Therefore, even if an entire population is replaced, the genetic algorithm will remember the best solutions found.

The algorithm continues these operations until the technique has converged on the best solution at which time each solution in the population will have basically the same bit make-up.

A weakness of genetic algorithms is its handling of constraints. Constraint violations are penalized in an objective function. The appropriate form of these constraint penalty functions can be difficult to determine; this is particularly true in this research.

2.6 GENESIS

Genetic Search Implementation System, GENESIS, is a program written in C-code that performs all the genetic algorithm operations (Grefenstette, 1990). All that is required is for the user to write his own evaluation function which determines a candidate solution's objective function value.

The user determines how many generations the algorithm should generate as well as the size of the initial population. The user also determines how to generate the initial solutions. It can be done randomly, or the initial population can be 'seeded' with solutions found by a greedy heuristic. The user also determines which selection strategy to use, the crossover rate, and the mutation rate.

After this has been established, GENESIS sends the individual member of the current population to a user-defined program. The solution is sent in a bit stream to the evaluation function. When all the decision variables are binary variables, the length of the bit stream is equivalent to the number of variables. The user-defined evaluation function checks the feasibility of the solution and assigns penalties for constraint violations. After the program determines the objective function value (including penalties), the value is sent back to GENESIS. After all the members of the population have been evaluated, GENESIS performs all the operations necessary to produce a new generation. Once a new generation is produced, these solutions are again sent to the user-defined program. This procedure repeats itself until the program reaches the number of generations specified by the user. Figure 2-3 provides an illustration of what we have just described.

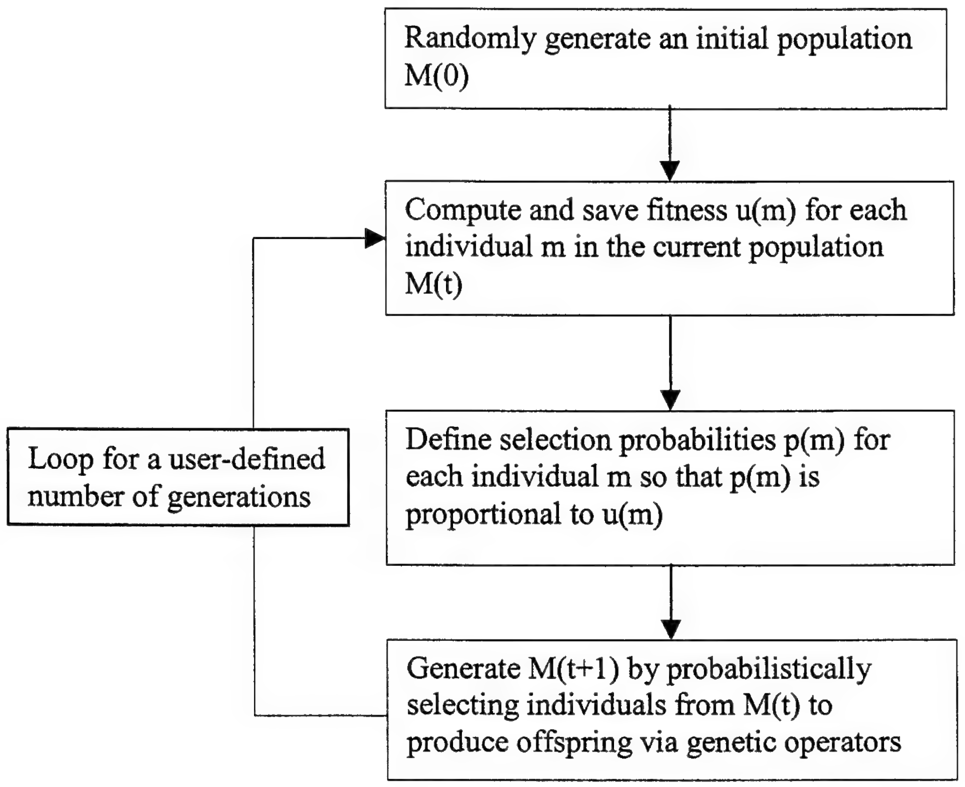
Randomly generate an initial population M(0)
Compute and save fitness u(m) for each individual m in the current population M(t)
Define selection probabilities p(m) for each individual m so that p(m) is proportional to u(m)
Generate M(t+1) by probabilistically selecting individuals from M(t) to produce offspring via genetic operators
Loop for a user-defined number of generations

Figure 2-3: Graphical Depiction of how GENESIS Operates

2.7 Summary

There has not been a large amount of research performed on the three-dimensional pallet-packing problem. However, the problem has gathered interest in the operations research community. Appendix A presents an annotated bibliography of papers pertinent to packing problems.

In the next chapter we describe in detail the development of the three-dimensional pallet-packing formulation, the constraints included in the formulation, and the heuristic used to test and expand the formulation to handle larger problems.

Chapter 3 – Results

3.1 Mathematical Formulation

The initial focus of this research was on the development of a strict three-dimensional mathematical formulation for the distributor's pallet-packing problem. We chose a 0-1, nonlinear model. The objective of the formulation is to minimize unused packing space. To determine the amount of unused space, we broke the available packing space on the pallet into cubic units. Since the dimensions for both the pallet and boxes are given in inches, the cubic units of the pallet are $(\text{inches})^3$. Therefore, the number of cubic units available for packing equals the volume of the pallet.

We want the formulation to mimic actual packing procedures, so constraints force the algorithm to generate solutions in the same manner as a loadmaster. Therefore, the formulation allows the boxes to be packed in any of their six orientations.

The constraints included in the model ensure boxes are packed with the correct volume and dimensions, the available packing volume is not exceeded, each box has a complete foundation on which to be packed, and at most only one box occupies each cubic unit on the pallet. Following is a descriptions of the variables used in the formulation.

E	=	The amount of unoccupied space on the pallet
L	=	The length of the pallet
W	=	The width of the pallet
H	=	The height of the pallet
N	=	The number of boxes to be packed
PV	=	The pallet volume available for packing

L_x	=	The length of box x; where x goes from 1 to N
W_x	=	The width of box x; where x goes from 1 to N
H_x	=	The height of box x; where x goes from 1 to N
V_x	=	The volume of box x; where x goes from 1 to N
A_x	=	The number of adjacent faces for each box x; where x goes from 1 to N (we explain this concept in detail following the presentation of the formulation)
B_x	=	Binary variable used to represent whether box x is packed or not; where x goes from 1 to N
		0 = Box x is not packed
		1 = Box x is packed
B_{ijkx}	=	Binary variable used to represent whether part of box x is packed in pallet location i, j, k; where x goes from 1 to N, where i goes from 1 to L, where j goes from 1 to W, where k goes from 1 to H
		0 = Box x is not packed in pallet location i, j, k
		1 = Box x is packed in pallet location i, j, k
PL_{xy}	=	Binary variable which counts the number of times when summing across the length of the pallet the summed value for box x equals the length of that box; where x goes from 1 to N, where y goes from 1 to (W*H)
		0 = Length of box x along length y does not equal L_x
		1 = Length of box x along length y does equal L_x
PW_{xy}	=	Binary variable which counts the number of times when summing across the width of the pallet the summed value for box x equals the width of that box; where x goes from 1 to N, where y goes from 1 to (L*H)
		0 = Width of box x along width y does not equal W_x
		1 = Width of box x along width y does equal W_x
PH_{xy}	=	Binary variable which counts the number of times when summing across the height of the pallet the summed value for box x equals the height of that box; where x goes from 1 to N, where y goes from 1 to (L*W)
		0 = Height of box x along height y does not equal H_x
		1 = Height of box x along height y does equal H_x

The formulation is the following:

Minimize E

Subject to the following constraint sets:

$$\left(\sum_{x=1}^{N} B_x * V_x\right) + E = PV \tag{1}$$

$$\sum_{x=1}^{N} B_{ijkx} \leq 1 \qquad \forall \text{ i from 1 to L, j from 1 to W, k from 1 to H} \tag{2}$$

$$\sum_{x=1}^{N} B_{ijkx} - B_{ij(k+1)x} \geq 0 \qquad \forall \text{ i from 1 to L, j from 1 to W, k from 1 to (H-1)} \tag{3}$$

$$\left(\sum_{i=1}^{L}\sum_{j=1}^{W}\sum_{k=1}^{H} B_{ijkx}\right) - B_x * V_x = 0 \qquad \forall \text{ x from 1 to N} \tag{4}$$

$$\left(\sum_{k=1}^{H}\sum_{j=1}^{W}\sum_{i=1}^{L-1} B_{ijkx} * B_{(i+1)jkx}\right) + \left(\sum_{k=1}^{H}\sum_{i=1}^{L}\sum_{j=1}^{W-1} B_{ijkx} * B_{i(j+1)kx}\right) + \left(\sum_{j=1}^{W}\sum_{i=1}^{L}\sum_{k=1}^{H-1} B_{ijkx} * B_{ij(k+1)x}\right) - A_x * B_x = 0 \qquad \forall \text{ x from 1 to N} \tag{5}$$

$$\sum_{i=1}^{L} B_{ijkx} = L_x * PL_{xy} \qquad \forall \text{ j from 1 to W, k from 1 to H, x from 1 to N, y from 1 to (W*H)} \tag{6a}$$

$$\sum_{y=1}^{W*H} PL_{xy} = W_x * H_x \qquad \forall \text{ x from 1 to N} \tag{6b}$$

$$\sum_{j=1}^{W} B_{ijkx} = W_x * PW_{xy} \qquad \forall \text{ i from 1 to L, k from 1 to H, x from 1 to N, y from 1 to (L*H)} \tag{6c}$$

$$\sum_{y=1}^{L*H} PW_{xy} = L_x * H_x \qquad \forall \text{ x from 1 to N} \tag{6d}$$

$$\sum_{k=1}^{H} B_{ijkx} = H_x * PH_{xy} \qquad \forall \text{ i from 1 to L, j from 1 to W, x from 1 to N, y from 1 to (L*W)} \tag{6e}$$

$$\sum_{y=1}^{L*W} PH_{xy} = L_x * W_x \qquad \forall \text{ x from 1 to N} \tag{6f}$$

B_{ijkx}, B_x, PL_{xy}, PW_{xy}, and $PH_{xy} =$ 0 or 1

The objective of this formulation minimizes the amount of unused packing space on the pallet. Constraint (1) ensures the volume of the packed boxes is less than the available volume that can be packed. Constraint (2) is a set of constraints ensuring no more than one box is placed in each cubic location on the pallet. Therefore, the number of constraints in this set is equal to L*W*H.

Constraint (3) ensures each box has a foundation on which to be placed. This helps ensure load stability. The number of constraints in this set is equal to L*W*(H-1). Constraint (4) ensures each packed box is packed with the correct volume. The number of constraints in this set is equal to the number of boxes.

Constraint (5) is the first of two constraints ensuring each box is packed with the correct dimensions. We broke down each box into cubic units, where the number of cubic units for a box is equal to the volume of the box. Then to ensure each box was packed with the correct dimensions, we developed a formula to determine how many adjacent faces exist for a box with a given length, width and height. Figure 3-1 illustrates what we define as adjacent faces.

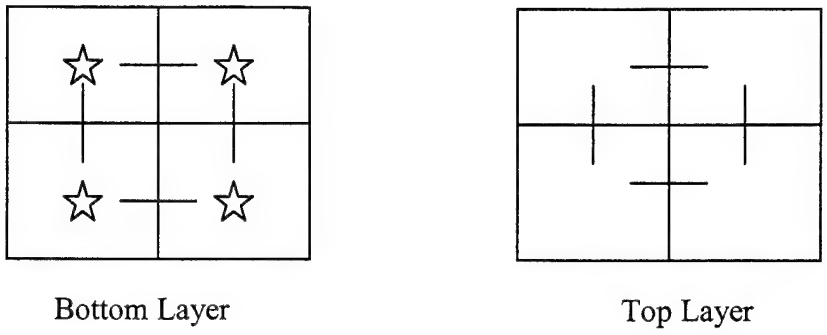

Figure 3-1: Box with Length, Width, and Height = 2

Figure 3-1 illustrates all eight cubic units of a box with length, width, and height dimensions of two. The lines intersecting the faces define the number of adjacent faces with regards to the length and the width of the box. For this box there are eight adjacent faces with regards to length and width. The stars represent the number of adjacent faces with regards to the height of the box, which in this case equals four. Therefore, there are 12 adjacent faces for a box with these dimensions.

Counting the adjacent faces of each box size is a tedious assignment, especially as box dimensions increase. Therefore, we developed an equation to determine the number of adjacent faces for a box with given dimensions. This equation is shown below.

L	=	*Length of Box*
W	=	*Width of Box*
H	=	*Height of Box*
A	=	*Number of adjacent Faces*

$$A = (L-1)(W*H)+(W-1)(L*H)+(H-1)(L*W) \tag{7}$$

The first term of equation (7) finds all of the adjacent faces in the direction of the length. The number of adjacent faces along the length of a box is always equal to the length of the box minus one unit of length. This value is multiplied by the width and the height of the box to determine the number of adjacent faces for the length dimension. The same method is applied to the width and the height of the box to determine the number of adjacent faces for each of those dimensions. The adjacent faces for each dimension are added together to determine the total number of adjacent faces for a box.

Unfortunately, ensuring a box is packed with the correct volume and the correct number of adjacent faces does not ensure a box is packed with the correct dimensions. The figure below illustrates the phenomenon when the original five constraints do not ensure a box is packed with the correct dimensions.

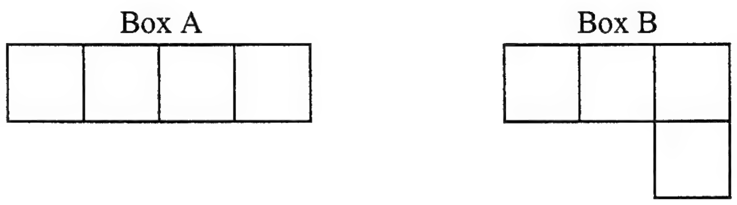

Figure 3-2: Ensuring Correct Box Dimensions

Figure 3-2 provides a depiction of two possible ways to pack a box with a volume of four and three adjacent faces. Box A is packed correctly while Box B is not. However in both cases the volume of the packing and the number of adjacent box faces are correct. Therefore, a sixth constraint is required to ensure all boxes are packed with the correct dimensions.

Constraint (6) is broken up into three different sections with two equations within each section. Each section represents a dimension. For example, equation (6a) illustrates that for each width and height pair, and for each box, the formulation sums over the length and counts how many units are occupied for each box. If the number of units equals the actual length of the box then the binary variable, PL_{xy}, is set to one, otherwise it is zero.

Equation (6b) ensures the correct length of each box occurs the correct number of times. The number of correct lengths for a box equals W_x*H_x. For a box with length, width, and height dimensions of two, there will be four times when the packed length of a box (for a given width and height) equals two.

The other two sets of equations (6c and 6d, and 6e and 6f) ensure both the width and the height of each box is correct throughout the pallet. For each box, equations (6a, 6c, and 6e) are performed $W*H$, $L*H$, and $W*L$ times respectively, while equations (6b, 6d, and 6f) are performed once for each box. The constraints in constraint sets (5) and (6) work together to ensure boxes are packed with the correct dimensions.

As the size of the problem increases, the number of variables and constraints required to formulate the problem becomes very large. The equations below illustrate the number of variables and constraints for a problem with N boxes and pallet dimensions L, W, and H.

$$Variables = N + (N*L*W*H) + (N*W*H) + (N*L*H) + (N*L*H) \tag{8}$$

$$Constraints = 1+5N+(L*W*H)+(L*W*(H-1))+(W*H*N)+(L*H*N)+(L*W*N) \tag{9}$$

3.2 Results from LINGO

For a small problem, we tested the formulation using HYPERLINGO, an optimization software package developed by LINDO Systems Inc. A small problem stays within the limits of the software, produces reasonable run times, and allows a validation of the model results (e.g., we can check the packing). The test case developed had three separate boxes. Table 3-1 presents the characteristics of each box.

Table 3-1: Box Characteristics for the Three Box Test Case

Box	Length	Width	Height	Volume	Adj. Faces
1	3	3	2	18	33
2	3	2	3	18	33
3	3	2	1	6	7

Additionally, the pallet the boxes were being packed on had a length, width, and height of three. Therefore, we knew the optimal solution was three, meaning that only three units of empty space should be left after the boxes were packed. Either box 1 or 2 would be packed and then box 3 would also be packed. This simple test case would allow us to determine if the constraints were functioning properly and if the formulation in general was performing as expected.

Without constraint (6), the boxes were packed with the correct dimensions. Therefore, to simplify the problem and to allow the boxes to be packed in any of their six orientations, we did not include constraint (6) in our HYPERLINGO formulation.

As with any nonlinear code, HYPERLINGO can only find locally optimum solutions as opposed to the global optimal solution. Therefore, we forced the program to pack either boxes one and three or boxes two and three to ensure it would at least follow the packing rules defined by the constraint sets provided in the previous section. The coded formulation and the outputted results, which illustrate the boxes were packed feasibly, are both shown in Appendix B.

3.3 Results from GENESIS

As the pallet dimensions and the number of boxes increase, the three-dimensional packing problem gets large and complex and finding a solution requires an unreasonable amount of time. To expand the size of the problem we applied a heuristic solution technique. We applied a genetic algorithm, using the GENESIS (Genetic Search Implementation System) software product.

Since the variables for this program are binary, the solution sent to the evaluation program by GENESIS has a length equal to the number of variables. For this problem, the number of variables is equal to the pallet volume times the number of boxes. For the simple three box test case, the number of variables sent from GENESIS to the evaluation function is equal to 81.

In order to measure the performance of the genetic algorithm, we applied the same test case we used in HYPERLINGO. Penalties were assigned for each constraint violation to get the genetic algorithm to converge near the optimal solution. Therefore each time a constraint was violated, a penalty was assigned. Testing showed the assignment of penalties was critical.

Appendix C shows the code for the evaluation function we developed. Table 3-2 shows the final penalties used for each constraint. Table 3-3 shows the genetic algorithm parameters used with GENESIS.

Table 3-2: Penalties for Constraint Violations

Constraint Violation	Penalty(Per violation)
Constraint 1	500
Constraint 2	500
Constraint 3	500
Constraint 4	100
Constraint 5	100
Constraint 6	100

Table 3-3: Parameter Setting for the Three Box Test Case

Parameter	Setting
Population size	50
Crossover rate	0.85
Mutation rate	0.01
Generation gap	0.9
Selection strategy	Elitist

Using the penalties from Table 3-2 and the parameter settings from Table 3-3, the genetic algorithm converged on the optimal solution in less than 200 generations. Therefore, a population size of 50 provided enough diversification to force the genetic algorithm to converge on the optimal solution. A crossover rate of 0.85 signifies that single-point crossover occurs on 85% of all pairs to be mated. A mutation rate of 0.01 indicates there is a 1% chance a bit will be mutated. Replacing 90% of parent solutions with offspring worked best for the generation gap. Lastly, we applied the elitist strategy to ensure the best characteristics are maintained from generation to generation. The complete input file and solution are shown in Appendix D.

The second test case, a 3 by 3 by 3 pallet with six different sized boxes, is shown in Table 3-4.

Table 3-4: Box Characteristics for the Six Box Test Case

Box	Length	Width	Height	Volume	Adj. Faces
1	2	2	2	8	12
2	3	2	1	6	7
3	1	2	2	4	4
4	2	1	1	2	1
5	1	3	1	3	2
6	1	1	1	1	0

Since we doubled the number of boxes to be packed, the length of the bit stream sent to the evaluation function also doubled. The bit stream consisted of 162 bits. Since the volume of all the boxes is 24 and the available packing volume is 27, the optimal solution for this problem packs all six boxes and leaves three empty units on the pallet.

We used the same penalties as those used in the three box problem, but modified the genetic algorithm parameters. Table 3-5 shows the parameter settings we used for this problem.

Table 3-5: Parameter Settings for the Six Box Test Case

Parameter	Setting
Population size	100
Crossover rate	0.95
Mutation rate	0.01
Generation gap	0.9
Selection strategy	Elitist

Since the solution space is larger for this problem, we increased the population size to 100. A crossover rate of 0.95 worked best for this problem. The other parameter settings remained the same as those used with the three box test problem. The entire input file and solution are presented in Appendix D. Using the penalties from Table 3-2 and the parameter settings in Table 3-5, the genetic algorithm converged on a solution 75% of optimal in 655 generations. The algorithm packed all boxes except for Box 2. Figure 3-3 shows how the boxes were packed on the pallet. A number inside of a square represents that a part of the numbered box is located in that pallet location.

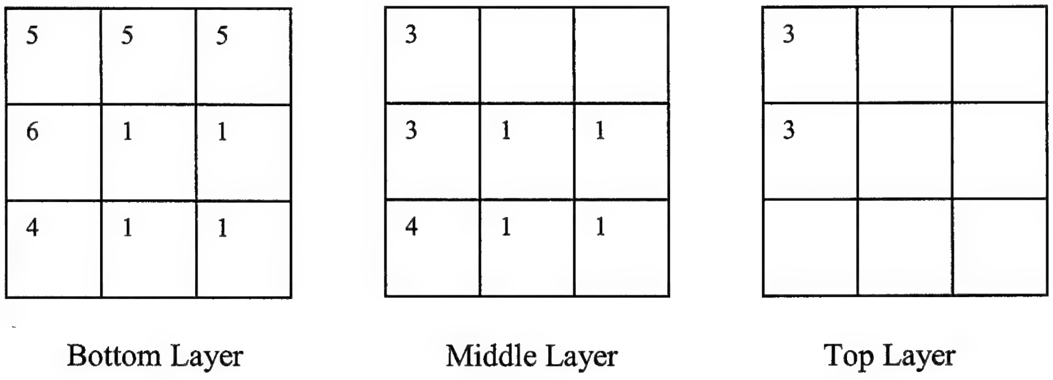

Figure 3-3: Packing of the Six Box Test Case

Our third test problem came from an article by Abdou and Arghavani (1996). The problem consists of a much larger pallet and eleven boxes. The length of the pallet is seven units while the width and height are four units. The characteristics of the eleven boxes are shown in Table 3-6.

Table 3-6: Box Characteristics for the Eleven Box Test Case

Box	Length	Width	Height	Volume	Adj. Faces
1	2	2	1	4	4
2	2	2	1	4	4
3	2	2	2	8	12
4	3	2	1	6	7
5	3	2	2	12	20
6	3	2	2	12	20
7	3	2	3	18	33
8	4	2	1	8	10
9	4	2	1	8	10
10	4	2	2	16	28
11	4	2	2	16	28

For this problem, Abdou and Arghavani (1996) were able to pack the pallet with 100% utilization. The length of the bit stream for this problem was 1,232 bits. Again we used the same penalties, but modified the parameter settings. Table 3-7 shows the parameter settings used for this problem.

Table 3-7: Parameter Settings for the Eleven Box Test Case

Parameter	Setting
Population size	100
Crossover rate	0.85
Mutation rate	0.01
Generation gap	0.9
Selection strategy	Elitist

Unfortunately, GENESIS did not show any signs of convergence in a reasonable amount of time. The reason for this is that GENESIS only allows for single-point crossover, which is too simplistic for a problem with this many variables. Even after 1,000 generations, which took approximately 45 minutes, the best solution did not even come close to resembling a feasible packing for any of the 11 boxes. Appendix D shows the complete input file as well as the results found after 1,000 generations. Although single-point crossover was effective for the smaller problem sizes, it was ineffective for a problem of this magnitude.

Even for the smaller problem sizes, multi-point crossover is preferred because it means more information is exchanged between the parents which leads to quicker convergence. We speculate that one potential method for determining the number of crossover points is to base it on the number of boxes.

3.4 Summary

We have developed a mathematical formulation for the three-dimensional pallet-packing problem and verified the formulation packs boxes correctly in three dimensions. Additionally, we successfully applied a genetic algorithm to our formulation. In Chapter Five we provide a formal conclusion for this research and recommendations for future work.

Chapter 4 – Conclusions and Recommendations

4.1 Introduction

We began this research with two objectives. The first objective was to develop a direct three-dimensional mathematical formulation for the pallet-packing problem. The second objective was to verify that the formulation correctly packed boxes onto a pallet. This verification included the development of a test problem to ensure the constraints caused the proper packing of the pallet.

In addition to meeting these objectives, we expanded the research to include the application of a heuristic solution approach. We attempted to use a genetic algorithm to determine if it was an effective tool which could be used to produce close-to-optimal feasible solutions.

In this chapter we discuss the results of the research and provide recommendations for future research in this area of study. We close this chapter with our final thoughts on this research topic.

4.2 Research Results

In our literature review of research accomplished on packing problems, we found only a minority of the research pertained to the three-dimensional pallet-packing problem.

Although most articles contained some of the necessary constraints for the Air Force's three-dimensional pallet-packing problem, none contained a formulation including all the required constraints. The article by Abdou and Arghavani (1996) was the most helpful in our initial formulation.

The objective of our formulation was to minimize the amount of unpacked space on the pallet. Three of the constraints included in the formulation ensure that each box is packed with the correct volume and the correct dimensions. The link boxes constraint as well as the box dimensions constraint work together to ensure each box is packed with the proper dimensions. To determine the number of adjacent faces for each box when it is broken down into cubic units, we developed a formula which calculates the number of adjacent faces among the unit cubes for each box based on that boxes length, width and height.

Two of the other constraints included in the formulation ensure each box has a foundation on which to be placed and that at most only one box can occupy each location on the pallet. The last constraint ensures the volume of all the packed boxes does not exceed the available pallet volume. Although these are not all the constraints required for the Air Force pallet-packing problem, using an optimal solver, we were able to show these constraints correctly pack boxes in three dimensions on a pallet.

Due to the complexity of the problem, we applied a genetic algorithm to our formulation. From the test cases, we concluded that genetic algorithms could be effectively used to find good feasible solutions to packing problems. Unfortunately, as the problem size increased, the solution quality decreased. We feel single-point crossover and the penalty functions are the two areas preventing the genetic algorithm from converging on better solutions for larger problems.

4.3 Recommendations for Future Research

Since we successfully applied a genetic algorithm to small test problems, further research should examine genetic algorithm application to larger problems. A genetic algorithm program which allows for multi-point crossover should be examined on larger more realistic problems to ultimately determine whether a genetic algorithm can find close-to-optimal feasible solutions for large, realistically sized problems.
If the genetic algorithm continues to provide poor results, either simulated annealing or tabu search should be investigated as possible heuristic solution techniques that could be applied to packing problems.

In addition to further examining heuristic solution techniques, the formulation should be expanded to include more of the necessary constraints required to pack Air Force 463L pallets. The first characteristic that must be included is the weight of the boxes. One constraint associated with weight ensures the total weight of all the boxes packed is less than the maximum allowable weight for the pallet. Another constraint ensures the heavier boxes are packed below the lighter boxes. A third constraint ensures each box is not crushed. This constraint ensures the total weight of all the boxes packed on top of a box does not exceed the maximum allowable weight that can be packed on that box. For safety reasons, a fourth constraint ensuring the center of gravity of the packed boxes is within four inches of the middle of the pallet is required by the Air Force. Another constraint to add, not pertaining to weight, ensures the top of the packing is as smooth as possible. Therefore, when the cargo net is thrown over the load, it will securely hold the boxes in place.

4.4 Final Thoughts

The Air Force has spent a lot of time and effort searching for a model which will more efficiently pack pallets. There is good reason for this as improved pallet utilization will cut costs and free additional time and resources. However, this is a very complex problem, and a heuristic solution approach is necessary to find a solution in a reasonable amount of time. Unfortunately, a major drawback of applying a heuristic is that it does not guarantee the optimal solution. However, it is the belief of the researcher that as more and more research is performed on heuristic solution approaches, a heuristic solution approach will be found for the three-dimensional pallet-packing problem that produces close-to-optimal, feasible solutions.

Appendix A – Annotated Bibliography

Abdou, G., and M. Yang. "A systematic approach for the three-dimensional palletization problem," <u>International Journal of Production Research 32</u>: 2381-2394 (1994).

The authors develop a multi-objective algorithm for the palletization problem. They attempt to maximize both pallet utilization and stability. Not many of the algorithms developed for this type of problem include stability as an objective. For the Air Force, stability is a necessity. The authors discuss the five factors that affect palletization: box size, box weight, box orientation, box density (stability), and box availability. They attempt to simplify the problem by assuming boxes with different base dimensions are grouped by the same height.

The authors develop blocks which are subsets of the actual pallet. They try to maximize the space within each block, by either filling it with a large box or with several smaller boxes. As expected, when new blocks are generated, the criteria is to use boxes that have not been previously selected, use the bigger boxes first, and lastly use as few boxes as possible. This is important since it will make it easy to pack the next layer. By packing the larger boxes first, either the volume of the block will be filled, or the larger boxes provide a good foundation for the boxes placed above them.

This algorithm has many features required by the Air Force. However, the authors note that expanding its capability so that boxes cannot by grouped by similar height and have different base dimensions is an area for improvement.

Abdou, G., and J. Arghavani. "Interactive ILP procedures for stacking optimization for the 3D palletization problem," International Journal of Production Research 35: 1287-1304 (1997).

The authors develop an algorithm that first maximizes the utilization of the base area and then attempts to maximize the stacking height of each sub area. The authors note that a limitation of this problem is high computational times, which limits the complexity of the models. Thus, the authors attempt to expand on a two-dimensional model so that it can be applied in the third dimension.

The authors acknowledge future research needs to be done to increase the capability of their model. The model does not address stability, and the model does not allow for a stochastic arrival of boxes with different dimensions. Lastly, possible modifications or better rules may be applied to the algorithm to aid in the heuristic procedure used.

Askin, Ronald G., and Charles R. Standridge. Modeling and Analysis of Manufacturing Systems. New York: John Wiley and Sons, Inc., 1993. (320-321)

The author describes two types of pallet packing problems. He first details the Manufacturer's pallet packing problem. This problem deals with loading boxes of the same dimensions. He next describes the Distributor's pallet-packing problem. This is more complex and involves boxes of different dimensions. This is the problem facing the Air Force and as the author states is very similar to the bin-packing and cutting stock problems.

"Astrokettle Algorithms."http://www4.bcity.com/astrokettle/data.html. 29 July 1999.

This article provides algorithms for the three-dimensional bin-packing problem. It also provides a demonstration for the bin-packing problem illustrating how effective their current algorithm is at finding a good solution. Again, there is a concern on how the program takes the weight of the boxes into account and how good of a solution is the program really finding. Thus, for the program to be useful for the Air Force, these questions need to be addressed.

Bischoff, E. E., F. Janetz, and M. S. W. Ratcliff. "Loading pallets with non-identical items," European Journal of Operational Research 84: 681-692 (1995).

The authors develop a three-dimensional heuristic approach to packing multiple-sized boxes onto a pallet. This problem is known as the 'Distributors Pallet Packing Problem.' The objective for their algorithm is to maximize pallet utilization while ensuring that the load is stable. Unlike the bin-packing problem where there are vertical walls ensuring the packing is stable, the packing on a pallet must be inherently stable to ensure the boxes don't topple. Additionally, the algorithm packs the boxes from the bottom upwards using single layers of up to two different box types at a time.

The algorithm attempts to pack the layers where vertical dimensions are the same. Additionally, the algorithm favors a layer in which one type of box can cover the entire layer. Also, larger gaps between boxes are favored over smaller ones in hopes a box might be able to fill the vacancy. The authors found that using the layering approach is beneficial in terms of achieving stability whereas other algorithms that attempt to pack columns are generally unstable. Also, with more box types the packing utilization decreased since it was more difficult for the algorithm to form layers with the boxes.

Overall I felt this was a good layered approach to the problem even though there was nothing more than an explanation of their algorithm. Lastly, the authors did add a merging criteria to their algorithm to prevent fragmentation. Even though this may have added to the content in the article it seemed to me to add very little to the quality of the solution found.

Bischoff, E. E., and M. S. W. Ratcliff. "Loading Multiple Pallets," Journal of the Operational Research Society 46: 1322-1336 (1995).

This paper focuses on the 'Distributors Pallet Packing Problem' and is mostly concerned with packing multiple pallets at a time. However there is a small portion of the article that is concerned with packing only one pallet at a time. Packing stability is addressed in this paper and the underlying algorithm used is the single-pallet algorithm by Bischoff. Unfortunately, this article packs the boxes on the pallet using a semi-layered approach. It tries to back boxes that have equivalent dimensions first to get them the same height. The author found when packing the multiple pallets sequentially as opposed to simultaneously the algorithm produced better utilization. He also found that with single-pallet packing a better utilization is found when there are fewer box sizes that need to be packed.

Chen, C. S., S. Sarin, and B. Ran. "The pallet packing problem for non-uniform box sizes," International Journal of Production Research 29: 1963-1968 (1991).

The authors develop an algorithm minimizing the number of pallets required to pack boxes of non-uniform size. The algorithm takes into account box orientation and allows the size of boxes to be non-integer. The formulation is for the two-dimensional case and according to the authors does find the optimal packing. The constraints in the formulation ensure there is no overlap among the boxes, every box is placed on the pallet, and that each box is placed within the confines of the pallet.

Dowsland, Kathryn A. "An exact algorithm for the pallet loading problem," <u>European Journal of Operational Research 31</u>: 78-83 (1987).

The author develops an algorithm for the 'manufacturers pallet loading problem.' The problem is reduced to the two dimensional problem of packing identical rectangles onto a larger containing rectangle. The author concedes that this problem is NP-complete and that most of the emphasis has been on finding good heuristic solution. The author uses a graph-theoretic model of the problem, but found that the problem becomes too large to use this method directly. She provides some problem reduction techniques allowing the problem to become more manageable. The information in this paper does not appear to be very applicable to the three-dimensional packing problem.

Dowsland, Kathryn A., and William B. Dowsland. "Packing problems," <u>European Journal of Operational Research 56</u>: 2-14 (1992).

Most of this article is spent describing the different types of packing problems and the constraints found in those problems. They begin with the two-dimensional rectangular packing problems. The authors state that the problem lends itself to dynamic programming, however they go on to state that the problem is generally formulated as an integer programming problem where there is one binary variable for each possible position for a piece on the pallet. They describe the pallet-loading problem, packing identical sized boxes, as well as the bin-packing problem. The bin-packing problem is similar to the pallet-packing problem however with the bin-packing problem there are vertical columns which provide vertical stability.

The authors then dive into the three-dimensional packing problem and state that the increased combinatorial complexity over the two-dimensional case means that exact solutions are unlikely to be found. Also, when moving to the third dimension load stability becomes an issue and has been found to be difficult to formulate. Lastly, due to the complexity of moving to the third dimension it becomes increasingly difficult to check the feasibility of a solution by eye.

Dowsland, Kathryn A. "Some experiments with simulated annealing techniques for packing problems," European Journal of Operational Research 68: 389-399 (1993).

The first part of the article is concerned with the classical pallet loading problem, packing identical rectangles into a larger containing rectangle. The second part of the article is concerned with the packing of non-identical pieces into a larger containing rectangle. For both parts the author was working on the two-dimensional case. Generally with the packing of identical items the objective function attempts to maximize the number of boxes packed, whereas with the packing on non-identical items the objective function attempts to minimize wasted space.

The paper describes the simulated annealing process for both types of problems in good detail making it pretty easy to follow and understand. Unfortunately for the non-uniform box size case the annealing process often produced infeasible solutions. The author found that after a good feasible solution was found the process would move away from that solution to a worse solution. The author concluded that simulated annealing could be a good solution technique for this problem if a few changes were made to the algorithm she developed.

Dowsland, William B. "Three-dimensional packing—solution approaches and heuristic development," International Journal of Production Research 29: 1673-1685 (1991).

Much of the work concerning the pallet-packing problem has been done in two dimensions. The author attempts to look at the third dimension and how heuristic methods can be improved to provide better results. Additionally, he looks at the possibility of expanding the two-dimensional heuristics so that they can be used in the three-dimensional packing problems. Unfortunately when looking at the third dimension the level of complexity increases dramatically and constraints such as load stability make it difficult to expand heuristics designed for the two-dimensional cases into the third dimension.

The author states that research on the three-dimensional packing problem is still in its early stages, thus most of the research states successful implementation, yet fails to show proof of scientific success. The author spends a lot of time discussing the wall building approach in which the pallet is filled in from the sides. However he discusses some of the troubles associated with this approach. The main concern was the lack of load stability which is a major concern of the Air Force. Additionally, he states that it is generally good packing practice to build a load up from the base of the pallet. Overall, this article provided good insight especially regarding the lack of transferability of the two-dimensional heuristics to the three-dimensional case.

Dowsland, W. B. "Improving palletisation efficiency—the theoretical basis and practical application," International Journal of Production Research 33: 2213-2222 (1995).

The basis of the article is the "manufacturers" pallet-loading problem in which the goal is to maximize the number of identical sized boxes onto a pallet. The goal of the paper was to determine the potential benefits of modifying the box sizes by some fixed percentage, either maintaining constant box volume or giving a slightly smaller box volume. The author tested this hypothesis with set percentage change, thus the paper does not answer the question, 'What volumetric change is needed to provide an improvement?' The research is interesting but does not pertain to the actual packing of the boxes onto the pallet.

Fuh-Hwa, F. Liu and C-J Hsiao. "A three-dimensional pallet loading method for single-size boxes," Journal of the Operational Research Society 48: 726-735 (1997).

The author addresses the packing of single-sized boxes onto a single pallet trying to maximize utilization while maintaining stability. It allows the boxes to be packed in any of the six orientations. However, this article uses the layered approach to pack the boxes. Thus once one box is packed a certain way, then the other boxes in that layer are also packed that same way so that they have the same height throughout the layer. The author does use LINDO to solve his problems.

In this paper the algorithm does not allow boxes to be packed if they will not be stable. Most algorithms that address stability wait until a layer is packed and then check to see if stability is met. If it is then good, if not then that packing is thrown out.

Han, Ching Ping, Kenneth Knott, and Pius J. Egbelu. "A heuristic approach to the three-dimensional cargo-loading problem," International Journal of Production Research 27: 757-774 (1989).

The authors develop a heuristic approach to the three-dimensional packing problem. This article is concerned with single-sized boxes. The packing is accomplished by packing the boxes so that one of the edges of each box is initially placed along a vertical edge of the pallet. Additionally, the third dimension is approached using the layered approach. Thus, the algorithm tries to find the correct orientation of the box to both maximize utilization and maximize the number of boxes packed.

Healy, Patrick, Marcus Creavin and Ago Kuusik. "An optimal algorithm for rectangle placement," Operations Research Letters 24: 73-80 (1999).

The author approached this problem as the cutting stock problem, when no overlapping is feasible. The bottom-left heuristic technique for placing boxes is used to increase the speed of the solver. This paper focuses on a two-dimensional problem and is closely related to the largest empty rectangular problem. Thus, after a box has been placed, depending on which bottom-left location is chosen, the program searches for the largest empty area and tries to find a box that will best fill that area.

This article emphasizes the two-dimensional case. However, the author claims this formulation can easily be extended into the third dimension. Instead of just sweeping the width and the depth of the boxes, one would also need the sweep the height. Again, the problem with this is the explosion of variables and complexity required to include the third dimension.

Herbert, Edward A. and Kathryn A. Dowsland. "A family of genetic algorithms for the pallet loading problem," Annals of Operations Research 63: 415-436 (1996).

The authors develop a generic algorithm that can be applied to the two-dimensional pallet-packing problem. The authors explain that this problem requires binary coding and is a NP-Hard problem. Their algorithm allows the initial packing to be infeasible and then works toward a feasible solution. Although the initial packing may not be feasible, the algorithm sufficiently penalizes this to drive the algorithm back to feasibility. The objective of the algorithm is to pack the maximum number of boxes onto the pallet while avoiding overlap. The authors conclude that although their algorithm is not able to compete with traditional heuristic solution methods, they do suggest that such an approach could prove fruitful for more complex problems.

Ivancic, N., Kamlesh Mathur, and Bidhu B. Mohanty. "An Integer Programming Based Heuristic Approach to the Three-dimensional Packing Problem," Journal of Manufacturing and Operations Management 2: 268-298 (1989).

The authors develop a three-dimensional packing algorithm. They are attempting to minimize cost, where cost is essentially equivalent to maximizing the utilization of the total packing. Their work focuses on loading multiple containers whereas my research is focused on the loading of a single pallet at a time. However, they do perform some test instances where they are only loading one container. To aid in the solution the authors employ a greedy heuristic search for the "biggest bang for the buck." Unfortunately to apply this heuristic requires knowledge of all the packing patters which can be computationally prohibitive. The heuristic is used in two steps: first determining a packing pattern for a container and then choosing the next box to pack.

Their algorithm is set up so that boxes of which there are more left and with smaller volume are more likely to be packed.

Manship, Wesley E., and Jennifer L. Tilley. A Three-Dimensional 364L Pallet Packing Model and Algorithm. MS thesis, AFIT/GIM/LAL/98S-3. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, September 1998.

The focus of this thesis is to explain the background of the pallet-loading problem and provide information on many of the existing models. They go into great detail on the background information and why this is such an important issue for the Air Force. They look at some of the different algorithms that have been used in attempting to solve this problem. In addition, the authors suggest a non-linear algorithm that can be used to assist in the pallet loading process.

The authors built a model consisting of three different components to handle the different areas of the pallet-packing problem. The first component dealt with hazardous cargo since they found that this often is a cause for re-packing the pallets. The next component of the model helps limit the candidate list of boxes that may fit on the pallet. This aids in the manageability of the problem. The last component of the model focuses on putting the actual boxes onto the pallet. They use a nonlinear objective function to load the boxes as tightly as possible. The authors mention one difficult aspect of the problem is determining with constraints and variables to include in the model. The models in this thesis provide feasible, but sub-optimal solutions.

Morabito, R. and S. Morales. "A simple and effective recursive procedure for the manufacturer's pallet loading problem," Journal of the Operational Research Society 49: 819-828 (1998).

The focus of the paper is the packing of uniform sized boxes, the "manufacturer's" problem. Additionally, the authors implemented the face-up principal which allows the boxes to be packed only one of two ways. Basically the problem is broken up into two smaller problems. The first problem is to determine how many boxes can be placed on a layer and then the second problem determines how many layers to pack. The authors also ignore all constraints related to weight, density, and fragility.

The authors use a depth-first tree search method to find the best packing for a particular problem. Only in 18 of the more than 20,00 problems was the algorithm unable to find the optimal packing. The authors developed two different algorithms to find the best packing. One of the algorithms was much quicker but did not perform as well on the whole.

"PowerPack^{TM} 6 Easy Steps to Lower Freight Costs." http://www.remarkable.co.nz/prod04.htm. 29 July 1999.

This article describes a three-dimensional packing software that allows for different size boxes with different attributes be loaded into the program. The program then determines the optimal packing of boxes onto the pallet to minimize empty space, and lastly prints the solution for the user. Some concerns I have with this product are how the weight of the boxes is factored into the problem and if there are limitations on the pallet size. Since the Air Force has weight limitations on its cargo and a center of gravity requirement, for a software package to be useful these things must be included. Secondly, most algorithms that attempt to solve this problem deal with pallets much smaller than what the Air Force uses, thus it is important this software is able to solve the problem using the 463L pallet size. Also, it does not say anything about the limitation on the number of boxes that can be loaded into the program for each problem. The cost of this software is only $995.

Pisinger, David. "David Pisinger's optimization codes." http://www.diku.dk/~pisinger/codes.html. 29 July 1999.

This article describes an algorithm in C-code that solves three-dimensional packing problems either as a heuristic or to optimality. The code as expected is quite lengthy, but could be used as a possible foundation in my thesis. One limitation for this code is that it does not allow the boxes to be rotated in any direction. Most algorithms allow boxes to be rotated 90° around the vertical axis, however by rotating the boxes, a lot of complexity is introduced. Thus, this may be one area where the code can be improved to increase its capabilities.

Romaine, Jonathan M. Solving the Multidimensional Multiple Knapsack Problem with Packing Constraints Using Tabu Search. MS thesis, AFIT/GOR/ENS/99M-15. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1999.

This thesis approaches the pallet-packing problem in two-dimensional space, however it allows for modifications to include the third dimension. Due to the solution time required to solve the problem the author employs Tabu search coded in JAVA to efficiently determine the best solution. Three approaches are used in loading the pallets. The first method is loading the pallets for the first plane before even thinking about the next plane. The next strategy is that each aircraft in the fleet is feasible and packable at the same time. The last strategy is that only one plane needs to be feasible and packable at a time. The last strategy provided the best solutions. Adding a third dimension to the model would make the model more realistic and allow one to take into account height restrictions.

Scheithauer, Guntram and Johannes Terno. "The G4-Heuristic for the Pallet Loading Problem," Journal of the Operational Research Society 47: 511-522 (1996).

The authors use this G4-heuristic to pack single-sized boxes in two dimensions. Thus this paper focuses on the 'Distributor's Problem'. They claim that this heuristic finds solutions at least as good as any other heuristic to date. Their presentation of their algorithm is very confusing and hard to follow. In addition I do not feel this article will be very helpful with my formulation since it is only concerned with two dimensions and it is only packing single-sized boxes.

Taylor, Gregory S. A Pallet Packing Postprocessor for the Logistics Composite Model. MS thesis, AFIT/GST/ENS/94M-11. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1994.

This paper provides an excellent description of the current problems with pallet packing in the Air Force and the needs for an algorithm that will enable the Air Force to better utilize pallet space to save money on airlift. The difficulty of the problem stems from the fact that this is the distributor's problem meaning there are multiple box sizes that need to be loaded. The author explains that the main constraining factor in the pallet-packing problem is the volume. Pallet space is usually used up before the weight constraint becomes a factor.

The author defines three models in the thesis. The objective function in the first model minimizes the maximum deviation in height of all the boxes. The second model both maximizes area coverage and minimizes deviation in heights. The third model, which was not tested in the paper, is based on model two except it minimizes the wasted volume for all of the packed boxes. The paper emphasizes the complexity of the problem. This paper does not take into account the center of gravity constraint, which requires a majority of the weight be in the center of the pallet.

Tsai, R.D., E.M. Malstrom, and W. Kuo. "Physical Simulation of a Three Dimensional Palletizing Heuristic," International Journal of Production Research 32: 1159-1171 (1994).

The authors develop a three-dimensional heuristic to maximize the pallet load. Since, this problem is NP-Hard a heuristic is imposed to reduce computational time so that a close to optimal solution will be reported. They model this problem with four different size boxes arriving randomly to the loading dock. Once to the loading dock the critical path method was used to determine the placement and sequence of the boxes. For example, for a box to be placed on another box there must be a solid, uniform foundation to place the box. It is not allowed to overhang.

The authors simplify the problem by only using four different box sizes. In addition, although the technique may be transferable to the problem I am attacking, the idea is different. The authors have developed this model to pack boxes arriving on a conveyor belt as opposed to loading boxes already at the loading site.

Tsai, R.D., E.M. Malstrom, and W. Kuo. "Three Dimensional Palletization of Mixed Box Sizes," IIE Transactions 25: 64-74 (1993).

The authors develop a model that generates exact optimal solutions in terms of volume utilization of the pallet. Due to computational limits this model is limited in the size of problems that it can solve. The authors do not place any restrictions on the box sizes, the pallet sizes, or the allowable number of different sized boxes. However, the model does not address the issue of load stability when determining where to pack the boxes. Additionally this model was developed for the individual pallet case, thus the objective is not to minimize the number of pallets required to fit the boxes. Lastly, the model only allows the boxes to be packed in one of two ways. They are allowed to rotate 90° around the vertical axis.

However, the model does ensure no two boxes on the pallet overlap, as well as ensure that each box is placed completely within the confines of the pallet. Once a solution has been found the model outputs the exact placement of a box on a pallet. Unfortunately since this is an NP-Hard problem as the number of boxes increases the computation time required to find the optimal solution increases limiting the practical use of this model.

Appendix B – LINGO Formulation

B.1 LINGO Formulation

To verify that the mathematical formulation we developed for the three-dimensional pallet-packing problem performed correctly, we input the formulation into LINGO. In this appendix we break down the formulation into segments and describe in detail the purpose of each segment. Then we provide the results of the formulation and describe in detail their significance. First, we have provided a list and description of each variable used in the formulation.

VOLUME(x)	=	The variable represents the volume for each of the three boxes. The volume of each box is initialized at the end of the program
PACK(x)	=	This is a binary variable which represents whether Box x is packed or not 0 = box is not packed 1 = box is packed
B(ijkx)	=	This is a binary variable which represents whether box x is packed in pallet location i, j, k where x, i, j, and k all go from 1 to 3 for this problem 0 = box x is not packed in location i, j, k 1= box x is packed in location i, j, k
E	=	The amount of unused pallet space after the boxes are packed

B.2 Model Initialization and Objective Function

This part of the formulation initializes all the variables and defines the number of boxes, and the length, width, and height of the pallet to be three. Since this is the beginning of the formulation a brief description of the model is presented within the code. After the variables have been initialized within the *SETS* routine, the objective function value is defined. It is obvious that we are attempting to minimize unpacked pallet space. Lastly, after the objective function value is defined we declare variable *PACK(x)* and variable *B(ijkx)* to be binary variables.

```
MODEL:

! Description: This formulation packs rectangular boxes;
! on a pallet in three dimensions;
! Both the variables B and Pack are binary variables;

SETS:
        BOXES/ 1..3/ : VOLUME,PACK;
        PALL / 1..3/;
        PALW / 1..3/;
        PALH / 1..3/;
        PACKAGE( PALL, PALW, PALH, BOXES): B;
ENDSETS

! The objective function attempting to minimize empty space;
! E is empty space;

[MIN] MIN = E;

 @FOR(BOXES(X): @BIN(PACK(X)));
@FOR(PACKAGE(I,J,K,X): @BIN(B(I,J,K,X)));
```

B.3 First and Second Constraints

Each of these constraints are well commented within the coding of the model. The first constraint is a nonlinear constraint which determines the amount of unpacked space on the pallet. The second constraint is actually a set of constraints where the number of constraints is equal to the number of boxes to be packed. For this problem, it is a set of three constraints. As the comment within the code describes the constraint ensures that each box is packed with the correct volume.

```
! First constraint attempts to find the amount of space left unpacked;
! This ensures the packed box volume is less than the available;
! pallet volume;

@SUM(BOXES(X): VOLUME(X)*PACK(X)) + E = 27;


! This set of constraints ensure that the boxes are;
! packed with the correct box volume;

@FOR( BOXES(X):
   @SUM( PALL(I):
      @SUM( PALW(J):
            @SUM( PALH(K):
                  B(I,J,K,X)
            )
         )
   ) = VOLUME(X)*PACK(X)
);
```

B.4 Third and Fourth Constraints

Again the function of each of these two constraints is commented into the code. The first constraint below is also a set of constraints where the number of constraints in the set is equal to the volume of the pallet. For this problem the number of constraints in this set is equal to 27. The second constraint of this page is also a set of constraints where the number of constraints in the set is equal to:

$L*W*(H-1)$	where	L	=	length of pallet
		W	=	width of pallet
		H	=	height of pallet

```
! This set of constraints ensure no two boxes occupy;
! the same space;

@FOR( PALL(I):
  @FOR( PALW(J):
       @FOR( PALH(K):
          @SUM( BOXES(X):
               B(I,J,K,X)
          ) <= 1
       )
  )
);

! This set of constraints ensure a box is packed on top of;
! another box;
! Each box must have a complete foundation for it to be packed;

@FOR ( PALL(I):
  @FOR ( PALW(J):
    @SUM( BOXES(X):
         B(I,J,1,X) - B(I,J,2,X) ) >= 0));

@FOR ( PALL(I):
  @FOR ( PALW(J):
    @SUM( BOXES(X):
         B(I,J,2,X) - B(I,J,3,X) ) >= 0));
```

B.5 Constraint Five

This constraint is well described by the comments within the code. The constraint for box 3 did not fit on this page and is located on the following page. Since LINGO does not converge on the globally optimal solution for nonlinear formulations we had to assign boxes two and three to the packing otherwise it did not come up with the globally optimal solution and we were unable to check if the constraints were being violated.

```
! The last set of constraints ensure that the boxes are packed;
! with the correct dimensions;
! This ensures that the number of adjacent faces for each;
! box is the correct number of adjacent faces for that box;

! This assigns boxes 2 and 3 to the packing;
!PACK( 1) = 1;
PACK( 2) = 1;
PACK( 3) = 1;

!Box 1;

@SUM (PALL(I):
  @SUM (PALW(J): B(I,J,1,1)*B(I,J,2,1)+B(I,J,2,1)*B(I,J,3,1) ))
+
@SUM (PALL(I):
  @SUM (PALH(K): B(I,1,K,1)*B(I,2,K,1)+B(I,2,K,1)*B(I,3,K,1) ))
+
@SUM (PALW(J):
  @SUM (PALH(K): B(1,J,K,1)*B(2,J,K,1)+B(2,J,K,1)*B(3,J,K,1) ))
- (PACK( 1)*33) = 0;

!Box 2;

@SUM (PALL(I):
  @SUM (PALW(J): B(I,J,1,2)*B(I,J,2,2)+B(I,J,2,2)*B(I,J,3,2) ))
+
@SUM (PALL(I):
  @SUM (PALH(K): B(I,1,K,2)*B(I,2,K,2)+B(I,2,K,2)*B(I,3,K,2) ))
+
@SUM (PALW(J):
  @SUM (PALH(K): B(1,J,K,2)*B(2,J,K,2)+B(2,J,K,2)*B(3,J,K,2) ))
-   (PACK( 2)*33) = 0;
```

```
!Box 3;

@SUM (PALL(I):
  @SUM (PALW(J): B(I,J,1,3)*B(I,J,2,3)+B(I,J,2,3)*B(I,J,3,3) ))
+
@SUM (PALL(I):
  @SUM (PALH(K): B(I,1,K,3)*B(I,2,K,3)+B(I,2,K,3)*B(I,3,K,3) ))
+
@SUM (PALW(J):
  @SUM (PALH(K): B(1,J,K,3)*B(2,J,K,3)+B(2,J,K,3)*B(3,J,K,3) ))
-   (PACK( 3)*7) = 0;
```

B.6 End of Program

As described in the comments this is where the volume of each box is initialized; right before the end of the program.

```
! Data required for the problem;

 DATA:
        VOLUME=  18, 18, 6;
ENDDATA
END
```

B.7 Results when Boxes One and Three are Packed

The output from LINGO when boxes one and three are packed is shown below. The objective function value is three which we know to be the optimal solution for this problem. Additionally, PACK(1) and PACK(3) are equal to one which reemphasizes that boxes one and three are packed in the solution while PACK(2) equals zero. Lastly, the long columns under each box illustrates where each box is packed. As we would expect the entire column is zero for Box 2. Also, we found Boxes 1 and 3 do not violate any constraints.

```
Objective value:                3.000000

               Variable      Value
               E             3.000000

               PACK( 1)      1.000000
               PACK( 2)      0.0000000
               PACK( 3)      1.000000

       Box 1                         Box 2                          Box 3
B( 1, 1, 1, 1)   0.0000000  B( 1, 1, 1, 2)   0.0000000  B( 1, 1, 1, 3)   1.000000
B( 1, 1, 2, 1)   0.0000000  B( 1, 1, 2, 2)   0.0000000  B( 1, 1, 2, 3)   1.000000
B( 1, 1, 3, 1)   0.0000000  B( 1, 1, 3, 2)   0.0000000  B( 1, 1, 3, 3)   1.000000
B( 1, 2, 1, 1)   0.0000000  B( 1, 2, 1, 2)   0.0000000  B( 1, 2, 1, 3)   1.000000
B( 1, 2, 2, 1)   0.0000000  B( 1, 2, 2, 2)   0.0000000  B( 1, 2, 2, 3)   1.000000
B( 1, 2, 3, 1)   0.0000000  B( 1, 2, 3, 2)   0.0000000  B( 1, 2, 3, 3)   1.000000
B( 1, 3, 1, 1)   0.0000000  B( 1, 3, 1, 2)   0.0000000  B( 1, 3, 1, 3)   0.0000000
B( 1, 3, 2, 1)   0.0000000  B( 1, 3, 2, 2)   0.0000000  B( 1, 3, 2, 3)   0.0000000
B( 1, 3, 3, 1)   0.0000000  B( 1, 3, 3, 2)   0.0000000  B( 1, 3, 3, 3)   0.0000000

B( 2, 1, 1, 1)   1.000000   B( 2, 1, 1, 2)   0.0000000  B( 2, 1, 1, 3)   0.0000000
B( 2, 1, 2, 1)   1.000000   B( 2, 1, 2, 2)   0.0000000  B( 2, 1, 2, 3)   0.0000000
B( 2, 1, 3, 1)   1.000000   B( 2, 1, 3, 2)   0.0000000  B( 2, 1, 3, 3)   0.0000000
B( 2, 2, 1, 1)   1.000000   B( 2, 2, 1, 2)   0.0000000  B( 2, 2, 1, 3)   0.0000000
B( 2, 2, 2, 1)   1.000000   B( 2, 2, 2, 2)   0.0000000  B( 2, 2, 2, 3)   0.0000000
B( 2, 2, 3, 1)   1.000000   B( 2, 2, 3, 2)   0.0000000  B( 2, 2, 3, 3)   0.0000000
B( 2, 3, 1, 1)   1.000000   B( 2, 3, 1, 2)   0.0000000  B( 2, 3, 1, 3)   0.0000000
B( 2, 3, 2, 1)   1.000000   B( 2, 3, 2, 2)   0.0000000  B( 2, 3, 2, 3)   0.0000000
B( 2, 3, 3, 1)   1.000000   B( 2, 3, 3, 2)   0.0000000  B( 2, 3, 3, 3)   0.0000000
```

```
B( 3, 1, 1, 1)     1.000000   B( 3, 1, 1, 2)     0.0000000  B( 3, 1, 1, 3)     0.0000000
B( 3, 1, 2, 1)     1.000000   B( 3, 1, 2, 2)     0.0000000  B( 3, 1, 2, 3)     0.0000000
B( 3, 1, 3, 1)     1.000000   B( 3, 1, 3, 2)     0.0000000  B( 3, 1, 3, 3)     0.0000000
B( 3, 2, 1, 1)     1.000000   B( 3, 2, 1, 2)     0.0000000  B( 3, 2, 1, 3)     0.0000000
B( 3, 2, 2, 1)     1.000000   B( 3, 2, 2, 2)     0.0000000  B( 3, 2, 2, 3)     0.0000000
B( 3, 2, 3, 1)     1.000000   B( 3, 2, 3, 2)     0.0000000  B( 3, 2, 3, 3)     0.0000000
B( 3, 3, 1, 1)     1.000000   B( 3, 3, 1, 2)     0.0000000  B( 3, 3, 1, 3)     0.0000000
B( 3, 3, 2, 1)     1.000000   B( 3, 3, 2, 2)     0.0000000  B( 3, 3, 2, 3)     0.0000000
B( 3, 3, 3, 1)     1.000000   B( 3, 3, 3, 2)     0.0000000  B( 3, 3, 3, 3)     0.0000000
```

B.8 Results when Boxes Two and Three are Packed

As expected the objective function value when Boxes 2 and 3 are packed equals 3. Also, we can see that PACK(1) equals zero while the other two boxes equal 1. Lastly, the columns for each box, illustrating where each box is packed, shows that we again have a feasible packing.

```
Objective value:              3.000000

              Variable        Value
                 E            3.000000

              PACK( 1)        0.0000000
              PACK( 2)        1.000000
              PACK( 3)        1.000000

           Box 1                        Box 2                              Box 3
B( 1, 1, 1, 1)  0.00000000   B( 1, 1, 1, 2)  1.000000     B( 1, 1, 1, 3)    0.0000000
B( 1, 1, 2, 1)  0.0000000    B( 1, 1, 2, 2)  1.000000     B( 1, 1, 2, 3)    0.0000000
B( 1, 1, 3, 1)  0.0000000    B( 1, 1, 3, 2)  1.000000     B( 1, 1, 3, 3)    0.0000000
B( 1, 2, 1, 1)  0.0000000    B( 1, 2, 1, 2)  1.000000     B( 1, 2, 1, 3)    0.0000000
B( 1, 2, 2, 1)  0.0000000    B( 1, 2, 2, 2)  1.000000     B( 1, 2, 2, 3)    0.0000000
B( 1, 2, 3, 1)  0.0000000    B( 1, 2, 3, 2)  1.000000     B( 1, 2, 3, 3)    0.0000000
B( 1, 3, 1, 1)  0.0000000    B( 1, 3, 1, 2)  1.000000     B( 1, 3, 1, 3)    0.0000000
B( 1, 3, 2, 1)  0.0000000    B( 1, 3, 2, 2)  1.000000     B( 1, 3, 2, 3)    0.0000000
B( 1, 3, 3, 1)  0.0000000    B( 1, 3, 3, 2)  1.000000     B( 1, 3, 3, 3)    0.0000000
```

B(2, 1, 1, 1)	0.0000000	B(2, 1, 1, 2)	1.000000	B(2, 1, 1, 3)	0.0000000
B(2, 1, 2, 1)	0.0000000	B(2, 1, 2, 2)	1.000000	B(2, 1, 2, 3)	0.0000000
B(2, 1, 3, 1)	0.0000000	B(2, 1, 3, 2)	1.000000	B(2, 1, 3, 3)	0.0000000
B(2, 2, 1, 1)	0.0000000	B(2, 2, 1, 2)	1.000000	B(2, 2, 1, 3)	0.0000000
B(2, 2, 2, 1)	0.0000000	B(2, 2, 2, 2)	1.000000	B(2, 2, 2, 3)	0.0000000
B(2, 2, 3, 1)	0.0000000	B(2, 2, 3, 2)	1.000000	B(2, 2, 3, 3)	0.0000000
B(2, 3, 1, 1)	0.0000000	B(2, 3, 1, 2)	1.000000	B(2, 3, 1, 3)	0.0000000
B(2, 3, 2, 1)	0.0000000	B(2, 3, 2, 2)	1.000000	B(2, 3, 2, 3)	0.0000000
B(2, 3, 3, 1)	0.0000000	B(2, 3, 3, 2)	1.000000	B(2, 3, 3, 3)	0.0000000
B(3, 1, 1, 1)	0.0000000	B(3, 1, 1, 2)	0.0000000	B(3, 1, 1, 3)	0.0000000
B(3, 1, 2, 1)	0.0000000	B(3, 1, 2, 2)	0.0000000	B(3, 1, 2, 3)	0.0000000
B(3, 1, 3, 1)	0.0000000	B(3, 1, 3, 2)	0.0000000	B(3, 1, 3, 3)	0.0000000
B(3, 2, 1, 1)	0.0000000	B(3, 2, 1, 2)	0.0000000	B(3, 2, 1, 3)	1.000000
B(3, 2, 2, 1)	0.0000000	B(3, 2, 2, 2)	0.0000000	B(3, 2, 2, 3)	1.000000
B(3, 2, 3, 1)	0.0000000	B(3, 2, 3, 2)	0.0000000	B(3, 2, 3, 3)	1.000000
B(3, 3, 1, 1)	0.0000000	B(3, 3, 1, 2)	0.0000000	B(3, 3, 1, 3)	1.000000
B(3, 3, 2, 1)	0.0000000	B(3, 3, 2, 2)	0.0000000	B(3, 3, 2, 3)	1.000000
B(3, 3, 3, 1)	0.0000000	B(3, 3, 3, 2)	0.0000000	B(3, 3, 3, 3)	1.000000

Appendix C – Genetic Algorithm Evaluation Function

C.1 Description of Evaluation Function and Variables used in Program

In this appendix we provide the evaluation function that we coded using the programming language C. GENESIS calls the evaluation function and sends it a solution. The evaluation function determines the objective function value of that solution and the penalties that should be assessed on that solution before returning the objective function value including any penalties that were assessed. There are comments throughout the code explaining what the code is doing. This code is written for the simple three box test case. Below is a list and description of the variables used in the program.

GENESIS sends these four variables to the evaluation function:

char str[];	This variable represents the actual 1s and 0s found in the solution.
int length;	This variable represents the length of the bit stream.
double vect[];	Neither of these two variables are used in this
int genes;	program since all variables are binary.

The following variables are all used within the evaluation function:

int box_vol[3] = {18, 18, 6};	Actual volumes for each box
int box_L[3] = {3, 3, 3};	Actual lengths for each box
int box_w[3] = {3, 2, 2};	Actual widths for each box
int box_h[3] = {2, 3, 1};	Actual heights for each box
int adj_faces[3] = {33, 33, 7};	Number of adjacent faces for each box

int num_boxes = 3;	Number of boxes to be packed
int pal_L = 3;	Length of the pallet
int pal_w = 3;	Width of the pallet
int pal_h = 3;	Height of the pallet
int pal_vol = 27;	Available packing volume on the pallet

int box_pal[3][3][3][3];	Four dimensional array which holds the solution sent by GENESIS. The first dimension is for the boxes, and the last three dimensions are for the length, width, and height of the pallet

Each of the following constraints are used to check whether a solution sent by GENESIS violates a certain constraint.

int pack_pal[3][3][3];	This checks to see whether only one box is placed at each location
int pack_pal_mod[3][3][3];	This checks whether each box has a foundation on which to be packed
int count_adj_faces[3];	This checks whether each box has the correct number of adjacent faces
int sum_box[3];	This checks whether each box is packed with the correct volume
int wrong_L[3];	This checks whether each box is packed with the correct length
int wrong_w[3];	This checks whether each box is packed with the correct width
int wrong_h[3];	This checks whether each box is packed with the correct height
int violation[3];	This checks whether each box is packed with the correct dimensions
int mistake[3];	This counts how much the packed volume for each box differs from the actual volume of the box

Each of these constraints ensure that for each time the evaluation function is called the penalties are reset to zero.

int penalty1 = 0;	Checks whether boxes have correct volume
int penalty2 = 0;	Checks whether available pallet volume is exceeded
int penalty3 = 0;	Checks whether more than one box is packed at each location
int penalty4 = 0;	Checks whether each box has a foundation
int penalty5 = 0;	Checks whether each box has the correct number of adjacent faces
int penalty6 = 0;	Checks whether boxes are packed with the correct dimensions
int total_penalty = 0;	Total of all the above penalties

int same[3];	This variable checks whether the dimensions of each individual box are the same
int done;	Used to determine which orientation box was packed in
int match = 0;	Also used to determine which orientation box was packed in

Each of the following constraints are used to help determine if there are constraint violations.

int count_L = 0;	Used to ensures boxes are packed with correct length
int count_w = 0;	Used to ensure boxes are packed with correct width
int count_h = 0;	Used to ensure boxes are packed with correct height
int pack_vol = 0;	Used to ensure the available pallet volume is not exceeded
char temp[2];	This variable enables us to change the character string to an integer string
double obj_fun_val = 0;	Value returned to GENESIS after all penalties have been assessed

C.2 Function Preventing Negative Penalties

This function is called twice within the evaluation function to ensure no negative penalties are assessed.

```
/* This function ensures we do not get negative penalties */

int no_negative(int x, int y)
{
   if ((x - y) >= 0)
      return (x - y);
   else
      return (y - x);
}
```

C.3 Initialization of Variables

```
/* These are the variables which are passed in by Genesis */

double eval (str, length, vect, genes)
char str[];
int length;
double vect[];
int genes;
{
int pack_pal[3][3][3];
int pack_pal_mod[3][3][3];
int box_vol[3] = {18, 18, 6};
int box_L[3] = {3, 3, 3};
int box_w[3] = {3, 2, 2};
int box_h[3] = {2, 3, 1};
int adj_faces[3] = {33, 33, 7};
int count_adj_faces[3];

int penalty1 = 0;
int penalty2 = 0;
int penalty3 = 0;
int penalty4 = 0;
int penalty5 = 0;
int penalty6 = 0;
int done;
int total_penalty = 0;

int sum_box[3];
int wrong_L[3];
int wrong_w[3];
int wrong_h[3];
int violation[3];
int mistake[3];
int same[3];
int box_pal[3][3][3][3];

int num_boxes = 3;
int pal_h = 3;
int pal_L = 3;
int pal_w = 3;
int pal_vol = 27;

int match = 0;
int count_L = 0;
int count_w = 0;
int count_h = 0;
double obj_fun_val = 0;
int pack_vol = 0;

int i,H,W,L;

char temp[2];

for (H = 0; H < pal_h; H++)
   for (W = 0; W < pal_w; W++)
      for (L = 0; L < pal_L; L++)
      {
           pack_pal[L][W][H] = 0;
           pack_pal_mod[L][W][H] = 0;
      }


for (i = 0; i < num_boxes; i++)
{
  sum_box[i] = 0;
  count_adj_faces[i] = 0;
  wrong_L[i] = 0;
  wrong_w[i] = 0;
  wrong_h[i] = 0;
  violation[i] = 0;
  mistake[i] = 0;
  same[i] = 0;
}
```

C.4 Determine Whether Each Boxes Length, Width and Height are the Same

This is used in the constraint which checks whether each box is packed with the correct dimensions.

```
for (i = 0; i < num_boxes; i++)
{
    if ((box_L[i] == box_w[i]) && (box_L[i] == box_h[i]))
        same[i] = 3;
    else
        same[i] = 0;
}
```

C.5 Convert Character String to Binary String

This converts the character string into a binary string and puts the binary string into a four dimensional array. The first dimension represents the boxes, and the other three dimensions represent the length, width and height of the pallet respectively. Additionally, within the loop the program calculates the packed volume of each box in the solution as well as determine how many boxes are placed in each location.

```
/*  Must read in solution and put that solution into matrix form */

for ( i = 0; i < num_boxes; i++)
   for ( H = 0; H < pal_h; H++)
      for ( W = 0; W < pal_w; W++)
         for ( L = 0; L < pal_L; L++)
         {
            temp[0] = str[pal_vol*i + pal_h*pal_w*H + pal_L*W + L];
               temp[1] = '\0';
            box_pal[i][L][W][H] = atoi(temp);
            sum_box[i] = sum_box[i] + box_pal[i][L][W][H];
            pack_pal[L][W][H] = pack_pal[L][W][H] + box_pal[i][L][W][H];
         }
```

C.6 First Feasibility Check

This checks whether the packed volume of each box is equal to the actual volume of that box. If there is a violation then a penalty is assessed.

```
/* First feasibility check is to see if the volume of the boxes packed are the actual box
volume */


for (i = 0; i < num_boxes; i++)
   if ((sum_box[i] != box_vol[i]) && (sum_box[i] != 0))
            mistake[i] = mistake[i] + no_negative(box_vol[i], sum_box[i]);

for (i = 0; i < num_boxes; i++)
         penalty1 = penalty1 + mistake[i];
```

C.7 Second Feasibility Check

This checks whether the packed volume exceeds the available volume. If it does then a penalty is assessed.

```
/* Second feasibility check is to see if the volume of the solution is greater than the
available volume on the pallet */

for (i = 0; i < num_boxes; i++)
   pack_vol = pack_vol + sum_box[i];

if (pack_vol > pal_vol)
   penalty2 = pack_vol - pal_vol;
```

C.8 Third Feasibility Check

This checks whether more than one box occupies each location on the pallet and assesses a penalty for each violation.

```
/* Third feasibility check only allows one box to be placed in each location */

for (H = 0; H < pal_h; H++)
   for (W = 0; W < pal_w; W++)
      for (L = 0; L < pal_L; L++)
      {
         if (pack_pal[L][W][H] > 1)
            penalty3 = penalty3 + (pack_pal[L][W][H] - 1);
      }
```

C.9 Fourth Feasibility Check

This checks whether each box has a foundation on which to be packed and assigns a penalty for each location where the foundation does not exist.

```
/*  Fourth feasibility check only allows a box to be packed if there is a foundation for it */

for (H = 0; H < pal_h; H++)
   for (W = 0; W < pal_w; W++)
      for (L = 0; L < pal_L; L++)
      {
         if (pack_pal[L][W][H] > 0)
            pack_pal_mod[L][W][H] = 1;
         else
            pack_pal_mod[L][W][H] = 0;
      }

for (W = 0; W < pal_w; W++)
   for (L = 0; L < pal_L; L++)
         for (H = 1; H < pal_h; H++)
         {
            if ((pack_pal_mod[L][W][H] - pack_pal_mod[L][W][H-1]) > 0)
               penalty4 = penalty4 + 1;
         }
```

C.10 Fifth Feasibility Check

This checks whether the number of adjacent faces for each box in the solution is equal to the actual number of adjacent faces for that box. For each violation a penalty is assessed.

```
/* Fifth constraint ensures that the box is linked together */

for (i = 0; i < num_boxes; i++)
{
  for (H = 0; H < pal_h; H++)
  {
    for (W = 0; W < pal_w; W++)
    {
      for (L = 1; L < pal_L; L++)
      {
        if ((box_pal[i][L][W][H] == 1) && (box_pal[i][L-1][W][H] == 1))
          count_adj_faces[i] = count_adj_faces[i] + 1;
      }
    }
    for (L = 0; L < pal_L; L++)
    {
      for (W = 1; W < pal_w; W++)
      {
        if ((box_pal[i][L][W][H] == 1) && (box_pal[i][L][W-1][H] == 1))
          count_adj_faces[i] = count_adj_faces[i] + 1;
      }
    }
  }
  for (W = 0; W < pal_w; W++)
    for (L = 0; L < pal_L; L++)
      for (H = 1; H < pal_h; H++)
      {
        if ((box_pal[i][L][W][H] == 1) && (box_pal[i][L][W][H-1] == 1))
          count_adj_faces[i] = count_adj_faces[i] + 1;
      }
}

for (i = 0; i < num_boxes; i++)
                    if ((count_adj_faces[i] != adj_faces[i]) && (sum_box[i] > 0))
                     penalty5 = penalty5 + no_negative(adj_faces[i], count_adj_faces[i]);
```

C.11 Sixth Feasibility Check

This constraint checks to ensure each box is packed with the correct dimensions. It allows the box to be packed with any orientation. If the dimensions of a particular box are the same then the check is quite simple. Otherwise the check is quite intensive and consists of multiple loops to determine the orientation in which the box is packed. For each violation a penalty is assessed.

```
/* The sixth constraint ensures that each box is packed with the correct L, W, & H */

for (i = 0; i < num_boxes; i++)
{

  /* This first part is for boxes with all the same dimensions */

  if (same[i] == 3)
  {
    for (H = 0; H < pal_h; H++)
    {
      for (W = 0; W < pal_w; W++)
      {
        count_L = 0;
        for (L = 0; L < pal_L; L++)
          if (box_pal[i][L][W][H] == 1)
            count_L = count_L + 1;
        if ((count_L != box_L[i]) && (count_L != 0))
          wrong_L[i] = wrong_L[i] + 1;
      }

      for (L = 0; L < pal_L; L++)
      {
        count_w = 0;
        for (W = 0; W < pal_w; W++)
          if (box_pal[i][L][W][H] == 1)
            count_w = count_w + 1;
        if ((count_w != box_w[i]) && (count_w != 0))
          wrong_w[i] = wrong_w[i] + 1;
      }
    }
    for (W = 0; W < pal_w; W++)
    {
      for (L = 0; L < pal_L; L++)
```

```
        {
          count_h = 0;
          for (H = 0; H < pal_h; H++)
             if (box_pal[i][L][W][H] == 1)
                count_h = count_h + 1;
          if ((count_h != box_h[i]) && (count_h != 0))
             wrong_h[i] = wrong_h[i] + 1;
        }
      }
    }

/*  The rest of this is for boxes that do not have all three dimensions equal */
    else
    {
      match = 0;
      count_L = 0;
      done = 0;
      W = 0;
      H = 0;
      while (done == 0)
      {
        if ((W == (pal_w - 1)) && (H == (pal_h - 1)))
          done = 1;

        for (L = 0; L < pal_L; L++)
          if (box_pal[i][L][W][H] == 1)
             count_L = count_L + 1;

        W = W + 1;
        if (count_L > 0)
           done = 1;
        else if (W == pal_w)
        {
          H = H + 1;
          W = 0;
        }
        else
          {}
      }

      if (count_L == box_L[i])
        match = 1;
      else if (count_L == box_w[i])
        match = 2;
      else if (count_L == box_h[i])
        match = 3;
        else if (count_L != 0)
        {
          violation[i] = violation[i] + 1;
          match = 1;
        }
        else
          {}

/*  This is for boxes whose length is packed along the length of the pallet  */

        if (match == 1)
        {
          for (H = 0; H < pal_h; H++)
          {
            for (W = 0; W < pal_w; W++)
            {
              count_L = 0;
              for (L = 0; L < pal_L; L++)
                if (box_pal[i][L][W][H] == 1)
                  count_L = count_L + 1;
              if (( count_L != box_L[i]) && (count_L != 0))
                wrong_L[i] = wrong_L[i] + 1;
            }
          }

          done = 0;
          count_w = 0;
          L = 0;
          H = 0;
          while (done == 0)
          {
            if ((L == (pal_L - 1)) && (H == (pal_h - 1)))
              done = 1;

            for (W = 0; W < pal_w; W++)
              if (box_pal[i][L][W][H] == 1)
                count_w = count_w + 1;

            L = L + 1;
            if (count_w > 0)
              done = 1;
            else if (L == pal_L)
            {
              H = H + 1;
              L = 0;
            }
            else
                {}
        }

        if (count_w == box_w[i])
          match = 4;
        else if (count_w == box_h[i])
          match = 5;
        else if (count_w != 0)
        {
          violation[i] = violation[i] + 1;
          match = 4;
        }
        else
          {}

/* This is for boxes whose width is packed along the width of the pallet *
/
        if (match == 4)
        {
           for (H = 0; H < pal_h; H++)
           {
              for (L = 0; L < pal_L; L++)
              {
                 count_w = 0;
                 for (W = 0; W < pal_w; W++)
                    if (box_pal[i][L][W][H] == 1)
                       count_w = count_w + 1;
                 if ((count_w != box_w[i]) && (count_w != 0))
                    wrong_w[i] = wrong_w[i] + 1;
              }
           }
           for (W = 0; W < pal_w; W++)
           {
              for (L = 0; L < pal_L; L++)
              {
                 count_h = 0;
                 for (H = 0; H < pal_h; H++)
                    if (box_pal[i][L][W][H] == 1)
                       count_h = count_h + 1;
                 if ((count_h != box_h[i]) && (count_h != 0))
                    wrong_h[i] = wrong_h[i] + 1;
              }
           }
        }
```

```
/* This is for boxes whose height is packed along the width of the pallet */

         if (match == 5)
         {
           for (H = 0; H < pal_h; H++)
           {
             for (L = 0; L < pal_L; L++)
             {
               count_w = 0;
               for (W = 0; W < pal_w; W++)
                 if (box_pal[i][L][W][H] == 1)
                   count_w = count_w + 1;
               if ((count_w != box_h[i]) && (count_w != 0))
                 wrong_w[i] = wrong_w[i] + 1;
             }
           }
           for (W = 0; W < pal_w; W++)
           {
             for (L = 0; L < pal_L; L++)
             {
               count_h = 0;
               for (H = 0; H < pal_h; H++)
                 if (box_pal[i][L][W][H] == 1)
                   count_h = count_h + 1;
               if ((count_h != box_w[i]) && (count_h != 0))
                 wrong_h[i] = wrong_h[i] + 1;
             }
           }
         }
       }

/* This is for boxes whose width is packed along the length of the pallet */

       if (match == 2)
       {
         for (H = 0; H < pal_h; H++)
         {
           for (W = 0; W < pal_w; W++)
           {
             count_L = 0;
             for (L = 0; L < pal_L; L++)
               if (box_pal[i][L][W][H] == 1)
                 count_L = count_L + 1;
             if ((count_L != box_w[i]) && (count_L != 0))
               wrong_L[i] = wrong_L[i] + 1;
           }
```

```
        }

        done = 0;
        count_w = 0;
        L = 0;
        H = 0;
        while (done == 0)
        {
          if ((L == (pal_L - 1)) && (H == (pal_h - 1)))
            done = 1;

          for (W = 0; W < pal_w; W++)
            if (box_pal[i][L][W][H] == 1)
              count_w = count_w + 1;

          L = L + 1;
          if (count_w > 0)
            done = 1;
          else if (L == pal_L)
          {
            H = H + 1;
            L = 0;
          }
          else
            {}
        }

        if (count_w == box_L[i])
          match = 4;
        else if (count_w == box_h[i])
          match = 5;
        else if (count_w != 0)
        {
          violation[i] = violation[i] + 1;
          match = 4;
        }
        else
          {}

/* This is for boxes whose length is packed along the width of the pallet */

        if (match == 4)
        {
          for (H = 0; H < pal_h; H++)
          {
            for (L = 0; L < pal_L; L++)
            {
               count_w = 0;
               for (W = 0; W < pal_w; W++)
                  if (box_pal[i][L][W][H] == 1)
                     count_w = count_w + 1;
               if ((count_w != box_L[i]) && ( count_w != 0))
                  wrong_w[i] = wrong_w[i] + 1;
            }
         }
         for (W = 0; W < pal_w; W++)
         {
            for (L = 0; L < pal_L; L++)
            {
               count_h = 0;
               for (H = 0; H < pal_h; H++)
                  if (box_pal[i][L][W][H] == 1)
                     count_h = count_h + 1;
               if ((count_h != box_h[i]) && (count_h != 0))
                  wrong_h[i] = wrong_h[i] + 1;
            }
         }
      }

/*  This is for boxes whose height is packed along the width of the pallet  */

      if (match == 5)
      {
         for (H = 0; H < pal_h; H++)
         {
            for (L = 0; L < pal_L; L++)
            {
               count_w = 0;
               for (W = 0; W < pal_w; W++)
                  if (box_pal[i][L][W][H] == 1)
                     count_w = count_w + 1;
               if ((count_w != box_h[i]) && (count_w != 0))
                  wrong_w[i] = wrong_w[i] + 1;
            }
         }
         for (W = 0; W < pal_w; W++)
         {
            for (L = 0; L < pal_L; L++)
            {
               count_h = 0;
               for (H = 0; H < pal_h; H++)
                  if (box_pal[i][L][W][H] == 1)
```

```
                    count_h = count_h + 1;
                if ((count_h != box_L[i]) && (count_h != 0))
                  wrong_h[i] = wrong_h[i] + 1;
              }
            }
          }
        }

/*  This is for boxes whose height is packed along the length of the pallet  */

        if (match == 3)
        {
          for (H = 0; H < pal_h; H++)
          {
            for (W = 0; W < pal_w; W++)
            {
              count_L = 0;
              for (L = 0; L < pal_L; L++)
                if (box_pal[i][L][W][H] == 1)
                  count_L = count_L + 1;
              if ((count_L != box_h[i]) && (count_L != 0))
                wrong_L[i] = wrong_L[i] + 1;
            }
          }

          done = 0;
          count_w = 0;
          L = 0;
          H = 0;
          while (done == 0)
          {
            if ((L == (pal_L - 1)) && (H == (pal_h - 1)))
              done = 1;

            for (W = 0; W < pal_w; W++)
              if (box_pal[i][L][W][H] == 1)
                count_w = count_w + 1;

            L = L + 1;
            if (count_w > 0)
              done = 1;
           else if (L == pal_L)
           {
             H = H + 1;
             L = 0;
           }
```

```
      else
         {}
  }

      if (count_w == box_L[i])
         match = 4;
      else if (count_w == box_w[i])
         match = 5;
      else if (count_w != 0)
      {
         violation[i] = violation[i] + 1;
         match = 4;
      }
      else
         {}

/* This is for boxes whose length is packed along the width of the pallet */

      if (match == 4)
      {
         for (H = 0; H < pal_h; H++)
         {
            for (L = 0; L < pal_L; L++)
            {
               count_w = 0;
               for (W = 0; W < pal_w; W++)
                  if (box_pal[i][L][W][H] == 1)
                     count_w = count_w + 1;
               if ((count_w != box_L[i]) && (count_w != 0))
                  wrong_w[i] = wrong_w[i] + 1;
            }
         }
         for (W = 0; W < pal_w; W++)
         {
            for (L = 0; L < pal_L; L++)
            {
               count_h = 0;
               for (H = 0; H < pal_h; H++)
                  if (box_pal[i][L][W][H] == 1)
                     count_h = count_h + 1;
               if ((count_h != box_w[i]) && (count_h != 0))
                  wrong_h[i] = wrong_h[i] + 1;
            }
         }
      }
```

```
/* This is for boxes whose width is packed along the width of the pallet */
        if (match == 5)
        {
          for (H = 0; H < pal_h; H++)
          {
            for (L = 0; L < pal_L; L++)
            {
              count_w = 0;
              for (W = 0; W < pal_w; W++)
                if (box_pal[i][L][W][H] == 1)
                  count_w = count_w + 1;
              if ((count_w != box_w[i]) && (count_w != 0))
                wrong_w[i] = wrong_w[i] + 1;
            }
          }
          for (W = 0; W < pal_w; W++)
          {
            for (L = 0; L < pal_L; L++)
            {
              count_h = 0;
              for (H = 0; H < pal_h; H++)
                if (box_pal[i][L][W][H] == 1)
                  count_h = count_h + 1;
              if ((count_h != box_L[i]) && (count_h != 0))
                wrong_h[i] = wrong_h[i] + 1;
            }
          }
        }
      }
    }
}

for (i = 0; i < num_boxes; i++)
   if ((violation[i] > 0) || (wrong_L[i] > 0) || (wrong_w[i] > 0) || (wrong_h[i] > 0))
      penalty6 = penalty6 + violation[i] + wrong_L[i] + wrong_w[i] + wrong_h[i];
```

C.12 Objective Function Value

This calculates the total penalty associated with all the violations and then calculates the objective function value which is returned to GENESIS.

```
/*  This calculates the objective function value */

total_penalty = 50*penalty1 + 500*penalty2 + 500*penalty3 + 500*penalty4 +
100*penalty5 + 100*penalty6;

obj_fun_val = (pal_vol - pack_vol) + total_penalty;


return (obj_fun_val);

}
```

Appendix D – Results of Genetic Algorithm on Test Problems

D.1 Introduction

For each of these test problems we will show the genetic algorithms settings we used, the number of generations it took to find the reported solution, as well as provide the reported solution. We attempted to solve three separate test problems. The first test problem consisted of three boxes, the second test problem consisted of six boxes and the third test problem consisted of eleven boxes. Additionally, the available pallet volume for the first two test problems was 27, while for the third test problem the available pallet volume was 112.

D.2 Results of Test Problem One

The genetic algorithm settings we used for this problem are illustrated below.

```
      Experiments = 1
     Total Trials = 50000
  Population Size = 50
 Structure Length = 81
   Crossover Rate = .85
    Mutation Rate = .01
   Generation Gap = .9
   Scaling Window = 5
  Report Interval = 250
 Structures Saved = 10
Max Gens w/o Eval = 2
    Dump Interval = 0
     Dumps Saved = 0
          Options = cel
      Random Seed = 123456789
         Rank Min = 0.75
```

The settings we adjusted were the total trials (number of generations), population size, crossover rate, mutation rate, and generation gap. We found that these initial setting forced the genetic algorithm to converge on the optimal solution in less than 200 generations. The optimal solution is provided below. Each group of three bits represents a row, and each group of nine bits represents a layer of the pallet.

```
011 011 011 || 011 011 011 || 011 011 011    Box 1
000 000 000 || 000 000 000 || 000 000 000    Box 2
100 100 100 || 100 100 100 || 000 000 000    Box 3
```

Objective Function Value = 3.0000e+00

D.3 Results of Test Problem Two

For this test problem the size of the pallet remained the same as in the first test problem, however the number of boxes to be packed doubled. Therefore, the structure length of the bit stream also doubles from 81 variables to 162 variables. The input file used for this test problem is shown below.

```
      Experiments = 1
     Total Trials = 65000
  Population Size = 100
 Structure Length = 162
   Crossover Rate = .95
    Mutation Rate = .01
   Generation Gap = .9
   Scaling Window = 5
  Report Interval = 500
 Structures Saved = 10
Max Gens w/o Eval = 2
    Dump Interval = 0
      Dumps Saved = 0
          Options = cel
      Random Seed = 123456789
         Rank Min = 0.75
```

These were the settings we found to work best for this test problem. Unfortunately we were unable to get the genetic algorithm to converge on the optimal solution. Instead it converged on a sub-optimal feasible solution in 655 generations. The solution found is provided below.

```
011 011 000 || 011 011 000 || 000 000 000    Box 1
000 000 000 || 000 000 000 || 000 000 000    Box 2
000 000 000 || 000 100 100 || 000 100 100    Box 3
100 000 000 || 100 000 000 || 000 000 000    Box 4
000 000 111 || 000 000 000 || 000 000 000    Box 5
000 100 000 || 000 000 000 || 000 000 000    Box 6
```

Objective function value = 9.0000e+00

D.4 Results of Test Problem Three

This test problem is quite a bit larger than the other test problem and has a structure length of 1,232 variables. Therefore, it took a very long time for GENESIS to find solutions for this problem size. Additionally, since GENESIS only uses single-point crossover it did not come close to converging on a feasible solution in a reasonable amount of time. However, we did use the settings shown on the following page to solve the problem.

```
      Experiments = 1
     Total Trials = 100000
  Population Size = 100
 Structure Length = 1232
   Crossover Rate = .85
    Mutation Rate = .01
   Generation Gap = .9
   Scaling Window = 5
  Report Interval = 2000
 Structures Saved = 10
Max Gens w/o Eval = 2
    Dump Interval = 0
      Dumps Saved = 0
          Options = cel
      Random Seed = 123456789
         Rank Min = 0.75
```

Using these settings, GENESIS was unable to converge on a feasible solution. Also, it took GENESIS 45 minutes to run through all 1,000 generations. At the 99,907 iteration the best solution was found with an objective function value of 28,876.

Bibliography

Abdou, G., and M. Yang. "A systematic approach for the three-dimensional palletization problem," International Journal of Production Research 32: 2381-2394 (1994).

Abdou, G., and J. Arghavani. "Interactive ILP procedures for stacking optimization for the 3D palletization problem," International Journal of Production Research 35: 1287-1304 (1997).

Askin, Ronald G., and Charles R. Standridge. Modeling and Analysis of Manufacturing Systems. New York: John Wiley and Sons, Inc., 1993. (320-321)

"Astrokettle Algorithms."http://www4.bcity.com/astrokettle/data.html. 29 July 1999.

Beasley, D., D.R. Bull, and R.R. Martin. "An overview of genetic algorithms: Part 1, fundamentals," University Computing 15: 58-69 (1993).

Beasley, D., D.R. Bull, and R.R. Martin. "An overview of genetic algorithms: part 2, research topics," University Computing 15: 170-181 (1993).

Bischoff, E. E., F. Janetz, and M. S. W. Ratcliff. "Loading pallets with non-identical items," European Journal of Operational Research 84: 681-692 (1995).

Bischoff, E. E., and M. S. W. Ratcliff. "Loading Multiple Pallets," Journal of the Operational Research Society 46: 1322-1336 (1995).

Chen, C. S., S. Sarin, and B. Ran. "The pallet packing problem for non-uniform box sizes," International Journal of Production Research 29: 1963-1968 (1991).

Cheng, Runwei and Mitsuo Gen. Genetic Algorithms and Engineering Design. New York: John Wiley & Sons, Inc., 1997.

Chocolaad, Christopher A. Solving Geometric Knapsack Problems using Tabu Search Heuristics. MS thesis, AFIT/GOR/ENS/98M-05. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1998.

Computer Sciences Corporation. Unit Type Code Development, tailoring, and Optimization (UTC-DTO) Phase 2 Final Report. Contract DCA 100-94-D-00144. Falls Church VA: Defense Enterprise Integration Services, December 1997.

Dowsland, Kathryn A. "An exact algorithm for the pallet loading problem," European Journal of Operational Research 31: 78-83 (1987).

Dowsland, Kathryn A., and William B. Dowsland. "Packing problems," European Journal of Operational Research 56: 2-14 (1992).

Dowsland, Kathryn A. "Some experiments with simulated annealing techniques for packing problems," European Journal of Operational Research 68: 389-399 (1993).

Dowsland, William B. "Three-dimensional packing—solution approaches and heuristic development," International Journal of Production Research 29: 1673-1685 (1991).

Dowsland, W. B. "Improving palletisation efficiency—the theoretical basis and practical application," International Journal of Production Research 33: 2213-2222 (1995).

Fuh-Hwa, F. Liu and C-J Hsiao. "A three-dimensional pallet loading method for single-size boxes," Journal of the Operational Research Society 48: 726-735 (1997).

Grefenstette, John J. A User's Guide to GENESIS Version 5.0. 1990.

Han, Ching Ping, Kenneth Knott, and Pius J. Egbelu. "A heuristic approach to the three-dimensional cargo-loading problem," International Journal of Production Research 27: 757-774 (1989).

Healy, Patrick, Marcus Creavin and Ago Kuusik. "An optimal algorithm for rectangle placement," Operations Research Letters 24: 73-80 (1999).

Herbert, Edward A. and Kathryn A. Dowsland. "A family of genetic algorithms for the pallet loading problem," Annals of Operations Research 63: 415-436 (1996).

Ivancic, N., Kamlesh Mathur, and Bidhu B. Mohanty. "An Integer Programming Based Heuristic Approach to the Three-dimensional Packing Problem," Journal of Manufacturing and Operations Management 2: 268-298 (1989).

Kernighan, Brian W. and Dennis M. Ritchie. The C Programming Language 2nd Edition. New Jersey: Prentice-Hall, Inc., 1988.

Manship, Wesley E., and Jennifer L. Tilley. A Three-Dimensional 364L Pallet Packing Model and Algorithm. MS thesis, AFIT/GIM/LAL/98S-3. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, September 1998.

Morabito, R. and S. Morales. "A simple and effective recursive procedure for the manufacturer's pallet loading problem," Journal of the Operational Research Society 49: 819-828 (1998).

"PowerPack^{TM} 6 Easy Steps to Lower Freight Costs." http://www.remarkable.co.nz/prod04.htm. 29 July 1999.

Pisinger, David. "David Pisinger's optimization codes." http://www.diku.dk/~pisinger/codes.html. 29 July 1999.

Reeves, Colin R. Modern Heuristic Techniques for Combinatorial Problems. London: McGraw-Hill Book Company, 1995. (151-188)

Romaine, Jonathan M. Solving the Multidimensional Multiple Knapsack Problem with Packing Constraints Using Tabu Search. MS thesis, AFIT/GOR/ENS/99M-15. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1999.

Scheithauer, Guntram and Johannes Terno. "The G4-Heuristic for the Pallet Loading Problem," Journal of the Operational Research Society 47: 511-522 (1996).

TASC, Inc. (8 May 1998) Interim Project Report on the Feasibility of Finding Optimal Solutions to Three-Dimensional Packing Problems. Contract number F41624-97-D-5002, Air Force Research Labs/HESR, Wright-Patterson AFB OH.

Taylor, Gregory S. A Pallet Packing Postprocessor for the Logistics Composite Model. MS thesis, AFIT/GST/ENS/94M-11. Graduate School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, March 1994.

Tsai, R.D., E.M. Malstrom, and W. Kuo. "Physical Simulation of a Three Dimensional Palletizing Heuristic," International Journal of Production Research 32: 1159-1171 (1994).

Tsai, R.D., E.M. Malstrom, and W. Kuo. "Three Dimensional Palletization of Mixed Box Sizes," IIE Transactions 25: 64-74 (1993).

Vita

2Lt. Brian Ballew was born on 7, July 1976 in Panorama City, California. He grew up in Santa Clarita, California and eventually graduated from William S. Hart High School. He then attended the United States Air Force Academy in Colorado Springs, Colorado. On 27, May 1998 he received his commission as well as his Bachelors of Science degree in Operations Research and a minor in Mathematics. In August of 1998, he entered the Air Force Institute of Technology in pursuit of a Master's degree in Operations Research. His next assignment is to Shriever AFB, Colorado as a scientific analyst.